Oracle® Life Sciences Data Hub

Application Developer's Guide

Release 2.4.8
E95831-03
January 2020

Oracle Life Sciences Data Hub Application Developer's Guide, Release 2.4.8

E95831-03

Copyright © 2017, 2020, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Preface

Documentation accessibility xxv
Related resources xxv

1 Getting Started

About Application Development 1-1
Defining Objects 1-2
- Predefined Object Types 1-3
- Object Definitions and Instances, and their Containers 1-4
  - Organizational Objects 1-4
  - Object Definitions and Instances 1-5

Developing a Business Application 1-6
- Working in a Work Area 1-7
- Mapping Executables to Tables 1-8
- Example of a Business Application 1-9

Developing Standard Definitions and Modular Applications 1-12
Ensuring Data Currency 1-13

2 Creating Container Objects

Selecting a Domain 2-1
- Set a Default Domain in Preferences 2-1
- Select a Domain on the Applications Screen 2-2

Navigating in the Applications Tab 2-2
- Browsing the Main Applications Screen 2-2
- Navigating Using Breadcrumbs 2-4
- Using the Actions Drop-Down 2-4

Creating Domains, Application Areas, Work Areas, and Objects 2-4
- Creating Domains 2-5
  - Creating a Top-Level Domain 2-6
  - Creating a Child Domain 2-7
- Creating Application Areas 2-7

Creating Work Areas 2-8
Creating or Adding Children, including Object Instances 2-8
Creating Object Definitions 2-9
Modifying Domains, Application Areas, and Work Areas 2-9
Duplicating, Removing, Moving and Promoting Domains 2-10
Duplicating or Copying a Domain 2-10
Removing a Domain 2-11
Moving a Domain 2-11
Promoting a Domain 2-11
Managing Object Definitions 2-12
Copying Object Definitions 2-12
Moving Object Definitions 2-13
Creating Object Definitions 2-13

3 Common Development Tasks

Creating and Reusing Objects 3-2
Finding an Appropriate Definition 3-2
Reusing Existing Definitions 3-3
Creating an Instance of an Existing Definition 3-3
Using the Search and Select Window 3-4
Creating a New Definition and Instance 3-5
Creating and Using Object Descriptions 3-6
Naming Objects 3-7
Special Characters and Reserved Words 3-7
Duplicate Names: System Appends _1 3-8
Automatic Name Truncation 3-8
Keep Container and Object Names Short for Integrated Development Environments 3-8
Naming Rules for Specific Object Types 3-9
Naming Domains 3-9
Naming Source Code Objects 3-9
Naming Parameters 3-10
Naming OBIEE Business Areas 3-10
Customizable Naming Validation Package 3-10
Viewing Object Instances and Definitions 3-10
Viewing Object Instances 3-10
Viewing Object Definitions 3-11
Viewing Object Definitions in Domains and Application Areas 3-11
Viewing Object Definitions from Instances in a Work Area 3-11
Understanding Object Versions and Checkin/Checkout 3-12
About Object Versions 3-12

Checking Out Objects 3-13
Checking Out an Object Definition through an Instance 3-14
Checking Out an Object Definition Directly 3-15
Checking Out an Old Version of an Object Definition 3-15
Checking In Objects 3-15
Undoing Object Checkout 3-16
Version Labels 3-16
Version History 3-17
Versions of Component Objects 3-17
Upgrading Object Instances to a New Definition Version 3-18
Upgrading One or More Instances from the Definition 3-18
Upgrading to a Different Definition Version from an Instance 3-20
Upgrading to the Latest Version 3-20
Copying, Cloning, and Moving Objects 3-21
Copying Objects 3-21
Copying One Object Instance, Domain, Application Area, or Work Area 3-22
Copying One or More Objects at the Same Time 3-22
Copying an Old Version of an Object Definition 3-22
Comparison of Copying and Cloning Individual Objects 3-23
Cloning Objects 3-24
Comparison of Cloning Individual Objects and Whole Work Areas 3-25
Cloning Objects from the Application Development Screen 3-26
Cloning Objects from the Work Area Screen 3-26
Moving Objects 3-27
Pasting Objects 3-28
Removing Objects 3-28
Rules for Removing Objects 3-29
Classifying Objects and Outputs 3-30
About Classification 3-30
Classifying Objects 3-31
Classifying Outputs 3-32
Classifying Outputs Before They Are Generated 3-33
Classifying Outputs After They Are Generated 3-33
Applying Security to Objects and Outputs 3-34
Viewing User Group Assignments 3-34
Assigning User Groups to an Object 3-35
Removing User Group Assignments 3-35
Reassigning a User Group as Inherited 3-35
Validating Objects and Outputs 3-36
About Object Validation 3-36
About Output Validation 3-37

Validation Statuses 3-38
Changing Objects' Validation Status 3-38
Changing Outputs' Validation Status 3-39
Adding Supporting Information 3-39
Adding a Supporting Document 3-40
Adding a Supporting Output 3-40
Validation Rules 3-41
Reordering and Renumbering Objects 3-42
Defining and Mapping Table Descriptors 3-43
About Table Descriptors 3-43
Creating a Table Descriptor 3-44
Adding Source Table Descriptors 3-45
Adding Target Table Descriptors from a Remote Location 3-46
Adding Target Table Descriptors from a SAS File 3-46
Adding a Target Table Descriptor from an Existing Table Definition 3-47
Adding a Table Descriptor from a Metadata File 3-48
Adding a New Target Table Descriptor 3-49
Setting Table Descriptor Attributes 3-50
Adding or Uploading Columns 3-52
Mapping Table Descriptors to Table Instances 3-52
Automatic Mapping by Name 3-53
Mapping Table Descriptors Manually 3-54
Remapping 3-55
The Effects of Modifying the Table Descriptor or Table Instance on a Mapping 3-55
Mapping Columns 3-56
Mapping Columns Automatically 3-56
Mapping Columns Manually 3-56
Mapping Columns to Constants 3-58
Creating and Mapping Table Descriptors and Table Instances at the Same Time 3-60
Creating and Mapping Table Instances during Single Instance Installation 3-61
Creating Table Descriptors from Table Instances and Simultaneously Mapping Them 3-61
Creating Table Instances from Table Descriptors and Simultaneously Mapping Them 3-62
Unmapping Table Descriptors 3-63
Modifying Table Descriptors 3-64
Creating, Modifying, and Submitting Execution Setups 3-64
About Execution Setups and Templates 3-64
Creating an Execution Setup 3-67
Creating a New Execution Setup 3-67

Creating an Execution Setup from an Execution Template 3-69
Modifying an Execution Setup and Setting Parameters 3-69
Name and Description 3-70
Allow Use as Execution Template 3-70
System Parameters 3-70
Runtime Parameters 3-75
Report Set and Workflow Execution Setups 3-76
About Report Set and Workflow Execution Setups 3-76
Modifying a Report Set or Workflow Execution Setup 3-77
Removing an Execution Setup 3-79
Activating a Version of an Execution Setup 3-79
Submitting an Execution Setup 3-80
Viewing Data 3-80
Viewing Data within the Oracle Life Sciences Data Hub 3-81
About Data Browsing 3-81
Customizing Data Browsing 3-82
Viewing Data with Visualizations 3-85
Viewing Data with Program-Generated Reports 3-85
Viewing Data Through an IDE 3-85
Viewing Jobs 3-86
Viewing All Outputs of a Program or Report Set 3-87
Viewing All Outputs of an Instance in the Work Area Properties Screen 3-87
Viewing All Outputs from the My Home or Reports Tab 3-87
Viewing All Outputs of a Program or Report Set 3-87
Viewing All Program Outputs 3-87
Viewing All Report Set or Report Set Entry Outputs 3-88
Information Displayed 3-88
Using the Actions Drop-Down List 3-88
Defining Planned Outputs 3-90

4 Defining Tables

About Tables 4-1
Table Definitions and Instances 4-2
Reading from and Writing to Tables 4-2
Creating Tables 4-3
Columns 4-3
Reports on Table Definitions and Instances 4-3
Creating a Table 4-4
Creating a New Table Definition and Instance 4-5
Creating an Oracle Life Sciences Data Hub Table From a SAS Data Set 4-6

Creating a New Instance of an Existing Table Definition 4-7
Setting and Modifying Table Attributes 4-7
Process Type 4-7
Allow Snapshot 4-8
Blinding Flag 4-8
Blinding Status 4-9
Definition Source 4-9
Tablespace Name 4-9
Using the Table Properties Screen 4-10
Instance Properties 4-10
Definition Properties 4-11
Buttons 4-11
Defining Table Columns 4-12
Create a New Column and Variable 4-12
Create a Column from an Existing Variable 4-14
Defining Table Constraints and Indexes 4-14
About Constraints 4-14
Check Constraint 4-15
Non-Unique Index 4-16
Primary Key 4-16
Unique Key 4-17
Modifying Tables 4-17
Modifying Table Instance Properties 4-18
Modifying Table Definition Properties 4-19
Modifying Columns 4-19
Modifying Constraints and Indexes 4-20

5 Defining Programs

About Programs 5-2
Creating a Program 5-4
Creating a New Program Definition and Instance 5-4
Creating an Instance of an Existing Program Definition 5-6
Using the Program Properties Screen 5-6
Instance Properties 5-6
Definition Properties 5-7
Buttons 5-8
Defining Table Descriptors 5-8
Defining Source Code 5-9
About Source Code 5-10
Execution 5-11

Creating Source Code 5-11
Creating a New Source Code Definition and Instance 5-12
Creating an Instance of an Existing Source Code Definition 5-15
Calling APIs from Source Code 5-15
Creating and Using Static Reference Source Code 5-16
About Static Reference Source Code 5-16
Creating a Source Code for Use as a Static Reference 5-16
Using Static Reference Source Code 5-17
Upgrading Source Code And Undoing Source Code Upgrades 5-17
Upgrading a Single Source Code Instance 5-18
Upgrading Multiple Source Code Instances 5-20
Undoing Source Code Instance Upgrades 5-24
Defining Parameters 5-24
Defining Planned Outputs 5-25
About Planned Outputs 5-25
Defining a Planned Output 5-26
Planned Output Classification 5-27
Defining PL/SQL Programs 5-27
Writing Primary Source Code in PL/SQL 5-28
Testing PL/SQL Source Code 5-30
Creating a PL/SQL Package Storage Program 5-31
Using a Sharable PL/SQL Package 5-31
Compiling and Executing a PL/SQL Program 5-31
Manipulating Documents through a PL/SQL Program 5-32
Defining View Programs 5-32
Creating Source Code for a View Program 5-32
About View Table Instances 5-33
Blinding Status 5-33
About Table Descriptors in View Programs 5-34
Defining SAS Programs 5-34
SAS Program Development Process 5-35
Open SAS as an IDE from Oracle Life Sciences Data Hub 5-35
Upload Existing SAS Programs to Oracle Life Sciences Data Hub 5-35
Enter Source Code Directly in the Oracle Life Sciences Data Hub Source Code Definition 5-36
Connecting to SAS 5-36
SAS Program and Source Code Types 5-37
SAS Program 5-38
SAS Macro Catalog 5-38
SAS Format Catalog 5-39
Writing SAS Primary Source Code 5-39

Using a SAS Macro Catalog 5-41
Using a SAS Format Catalog 5-42
Creating a SAS Macro Catalog 5-42
Creating a SAS Macro Catalog 5-43
Nesting SAS Macros 5-43
About SAS Format Catalogs in the Oracle Life Sciences Data Hub 5-44
Creating a SAS Format Catalog 5-44
Calling an API to Capture Output Parameter Values 5-45
Defining Oracle Reports Programs 5-47
Defining Informatica Programs 5-47
Creating a New Informatica Program 5-48
Using Your Existing Informatica Mappings and Workflows 5-49
Creating and Synchronizing Source Code 5-49
Using PL/SQL Source Code in an Oracle Life Sciences Data Hub Informatica Program 5-50
Updating Table Descriptors 5-51
Setting Informatica Program Parameters 5-51
User-Defined Parameters 5-51
Predefined Parameters 5-51
Selective Index Management 5-52
Adding Planned Outputs 5-53
Informatica Integration 5-53
Informatica Folder Creation 5-53
Informatica Security Configuration 5-53
Defining Oracle Business Intelligence Publisher Programs 5-55
Integration with Oracle BI Publisher 5-55
Performing Oracle Life Sciences Data Hub Tasks 5-55
Performing Oracle BI Publisher Tasks 5-56
Running the Program 5-57
Editing an Existing Program 5-57
About Oracle BI Publisher Program Source Code 5-57
About Oracle BI Publisher Program Planned Outputs 5-58
Setting Oracle BI Publisher Program Parameters 5-58
Predefined Parameters 5-59
User-Defined Parameters 5-59
Installing Program Instances 5-59
Log File 5-60
IDE Launch Settings 5-60
About Launch Settings 5-61
Setting the Blind Break Value 5-62
Setting the Shared Snapshot Label Value 5-62

Modifying Programs 5-63
Modifying Program Instance Properties 5-63
Modifying Program Definition Properties 5-64
Modifying Table Descriptors 5-64
Modifying Source Code 5-65
Modifying Parameters 5-65
Modifying Planned Outputs 5-66
Setting Up Integrated Development Environments (IDEs) 5-66
Setting Up Oracle SQL Developer or SQL*Plus as an IDE 5-66
Launching Oracle SQL Developer 5-66
Relaunching Oracle SQL Developer from within Oracle Life Sciences Data Hub 5-67
Relaunching Oracle SQL Developer from outside Oracle Life Sciences Data Hub 5-68
Setting Up SAS as an IDE 5-68
Setting Up Informatica as an IDE 5-68

6 Defining Variables and Parameters

About Variables, Parameters, and Columns 6-1
Defining Variables 6-2
Creating Variables Automatically 6-2
Creating Variables through Load Sets 6-2
Creating Variables by Uploading SAS Data Sets and Variables during Table Definition 6-3
Creating Variables Manually 6-3
Modifying Variables 6-4
Using the Variable Properties Screen 6-5
Definition Properties 6-5
Buttons 6-6
Defining Columns 6-6
Defining Parameters 6-6
About Parameters 6-7
Using Parameters 6-7
Parameters and Execution Setups 6-8
Creating a Parameter 6-9
Create a New Parameter Definition and Instance 6-10
Create Parameters from an Existing Parameter Set 6-12
Defining Parameter Details 6-12
Setting a Method of Determining Value 6-12
Defining Allowed Values 6-13
Setting Validation Rules 6-14

Using the Parameter Properties Screen 6-15
About the Parameter Properties Screen 6-15
Instance Properties 6-15
Define Values 6-16
Validate Values 6-16
Definition Properties 6-17
Variable Properties 6-17
Buttons 6-17
Setting Up Parameter Value Propagation 6-17
About Parameter Value Propagation 6-18
Setting Up Value Propagation from the Source Parameter 6-19
Setting Up Value Propagation Manually 6-19
Setting Up Value Propagation Automatically 6-20
Setting Up Value Propagation from the Target Parameter 6-20
Defining and Using Parameter Sets 6-21
Explicitly Defining Parameter Sets 6-22
Defining Programatically Generated Lists of Values and Value Validation 6-22
Modifying Parameters 6-25
Modifying a Parameter Instance 6-26
Modifying a Parameter Definition 6-27
Modifying a Parameter Definition Directly in a Domain or Application Area 6-27
Modifying a Parameter Definition through a Parameter Set 6-28

7 Defining Load Sets

About Load Sets 7-1
Creating a Load Set 7-4
Creating a New Load Set Definition and Instance 7-5
Setting Load Set Attributes 7-5
Creating an Instance of an Existing Load Set Definition 7-5
Using the Load Set Properties Screen 7-6
Instance Properties 7-6
Definition Properties 7-6
Load Set Attributes 7-7
Buttons 7-7
Defining Table Descriptors 7-8
Setting Load Set Parameters 7-9
About Load Set Planned Outputs 7-10
Defining Different Load Set Types 7-10
Oracle Tables and Views 7-11
About Oracle Tables and Views Load Sets 7-11

Oracle Load Set Attributes 7-12
Oracle Load Set Table Descriptors 7-12
Oracle Load Set Parameters 7-12
Oracle Load Set Planned Outputs 7-13
Oracle Load Set Execution Setups 7-13
SAS 7-13
About SAS Load Sets 7-13
UTF8 Encoding 7-14
SAS Load Set Attributes 7-14
SAS Load Set Table Descriptors 7-14
SAS Load Set Parameters 7-15
SAS Load Set Planned Outputs 7-17
SAS Load Set Execution Setups 7-17
Text 7-18
About Text Load Sets 7-18
UTF8 Encoding 7-19
Text Load Set Attributes 7-19
Text Load Set Table Descriptors 7-19
Text Load Set Parameters 7-21
Text Load Set Planned Outputs 7-24
Text Load Set Execution Setup 7-25
Oracle Clinical Stable Interface 7-25
About Oracle Clinical Stable Interface Load Sets 7-26
OC Stable Interface Tables Load Set Attributes 7-26
OC Stable Interface Tables Load Set Table Descriptors 7-26
OC Stable Interface Tables Load Set Parameter 7-26
OC Stable Interface Tables Load Set Planned Outputs 7-26
OC Stable Interface Tables Load Set Execution Setups 7-26
Oracle Clinical Data Extract SAS Views 7-27
About OC DX SAS Views 7-27
OC DX SAS Views Load Set Attributes 7-27
OC DX SAS Views Load Set Parameters 7-27
OC DX SAS Views Load Set Table Descriptors 7-28
OC DX SAS Views Load Set Planned Outputs 7-28
OC DX SAS Views Load Set Execution Setups 7-28
Oracle Clinical Data Extract Views 7-28
About OC Data Extract View Load Sets 7-28
OC Data Extract Views Load Set Attributes 7-29
OC Data Extract Views Load Set Table Descriptors 7-30
OC Data Extract Views Load Set Parameters 7-30
OC Data Extract Views Load Set Planned Outputs 7-30

OC Data Extract Views Load Set Execution Setups 7-30
Oracle Clinical Design and Definition 7-30
About OC Design and Definition Load Sets 7-31
OC Design and Definition Load Set Attributes 7-31
OC Design and Definition Load Set Table Descriptors 7-31
OC Design and Definition Load Set Parameters 7-33
OC Design and Definition Load Set Planned Outputs 7-34
OC Design and Definition Load Set Execution Setups 7-34
Oracle Clinical Global Metadata 7-34
About OC Global Metadata Load Sets 7-34
OC Global Metadata Load Set Attributes 7-35
OC Global Metadata Load Set Table Descriptors 7-35
OC Global Metadata Load Set Parameters 7-35
OC Global Metadata Load Set Planned Outputs 7-35
OC Global Metadata Load Set Execution Setups 7-35
Oracle Clinical Labs 7-36
About OC Labs Load Sets 7-36
OC Labs Load Set Attribute 7-36
OC Labs Load Set Table Descriptors 7-36
OC Labs Load Set Parameters 7-37
OC Labs Load Set Planned Outputs 7-37
OC Labs Load Set Execution Setups 7-37
Oracle Clinical Randomization 7-37
About OC Randomization Load Sets 7-38
OC Randomization Load Set Attributes 7-39
OC Randomization Load Set Table Descriptor 7-39
OC Randomization Load Set Parameters 7-39
OC Randomization Load Set Planned Outputs 7-39
OC Randomization Load Set Execution Setups 7-39
Oracle Clinical Study Data 7-40
About OC Study Data Load sets 7-40
OC Study Data Load Set Attributes 7-40
OC Study Data Load Set Table Descriptors 7-40
OC Study Data Load Set Parameters 7-41
OC Study Data Load Set Planned Outputs 7-42
OC Randomization Load Set Execution Setups 7-42
Installing Load Set Instances 7-42
Access the Log File 7-43
Modifying Load Sets 7-43
Modifying Load Set Instance Properties 7-44
Modifying Load Set Definition Properties 7-44

Modifying Table Descriptors 7-45
Modifying Attributes and Parameters 7-45
Modifying Planned Outputs 7-45

8 Defining Data Marts

About Data Marts 8-2
Creating a Data Mart 8-2
Creating a New Data Mart Definition and Instance 8-3
Creating an Instance of an Existing Data Mart 8-3
Using the Data Mart Properties Screen 8-4
Instance Properties 8-4
Definition Properties 8-4
Data Mart Attributes 8-5
Setting Data Mart Attribute Values 8-5
Buttons 8-6
Defining Table Descriptors 8-6
Setting Data Mart Parameter Values 8-6
About Data Mart Planned Outputs 8-7
Defining Different Types of Data Marts 8-8
Defining Text Data Marts 8-8
About Text Data Marts 8-8
Text Data Mart Attributes 8-9
Text Data Mart Parameters 8-9
Delimited-Format Text Data Mart Parameters 8-10
Separator Character 8-11
Enclosing Character 8-11
Use Enclosing Character? 8-11
Text Data Mart Planned Outputs 8-11
Defining SAS Data Marts 8-12
About SAS Data Marts 8-12
SAS Data Mart Attribute 8-12
SAS Data Mart Parameters 8-13
SAS Data Mart Planned Outputs 8-13
Defining Oracle Export Data Marts 8-13
About Oracle Export Data Marts 8-13
Oracle Export Data Mart Attribute 8-14
Oracle Export Data Mart Parameters 8-14
Oracle Export Data Mart Planned Outputs 8-14
Installing Data Mart Instances 8-15
Access the Log File 8-15

Modifying Data Marts 8-16
Modifying Data Mart Instance Properties 8-16
Modifying Data Mart Definition Properties 8-17
Modifying Table Descriptors 8-17
Modifying Attributes and Parameters 8-17
Modifying Planned Outputs 8-18

9 Defining Report Sets

About Report Sets 9-2
How to Work on a Report Set 9-4
Creating Overlay Templates 9-6
About Overlay Templates 9-6
Creating Template Files 9-7
Creating an Overlay Template Definition 9-8
Creating an Overlay Template File Definition 9-8
Creating a Report Set 9-10
Creating a New Report Set Definition and Instance 9-11
Creating an Instance of an Existing Report Set Definition 9-12
Using the Report Set Properties Screen 9-12
Instance Properties 9-13
Definition Properties 9-13
Buttons 9-14
Using the Report Set Structure View 9-14
Navigating to the Report Set Structure View 9-15
Building and Modifying the Report Set 9-15
Modifying the Report Set 9-15
Report Set Entry Information Displayed 9-20
Creating and Setting Report Set Parameters 9-21
Setting Overlay Template Parameter Values 9-21
Overlay Template Parameters 9-22
Setting Overlay Template Parameters 9-22
OTD Parameter Value Sharing 9-23
Setting Post-Processing Parameter Values 9-24
Post-Processing Parameters at the Report Set Level Only 9-24
Post-Processing Parameters at the Report Set and Report Set Entry Levels 9-25
Setting Program Parameter Values 9-27
Creating Parameters for Sharing Values within the Report Set 9-27
Defining Report Set Entries 9-28
Creating Multiple Report Set Entries 9-28
Setting Report Set Entry Properties 9-32

Setting Overall Properties 9-32
Setting Title Properties 9-33
Setting Assigned Program Properties 9-33
Adding Narratives 9-34
Setting Numbering Properties 9-34
Setting Volume Break Properties 9-34
Adding Narratives 9-34
Defining Programs to Generate Reports 9-35
About Programs in Report Sets 9-35
Assigning a Planned Output to a Report Set Entry 9-38
Options from the Report Set Program View Screen 9-39
Viewing Planned Output Assignments 9-40
Passing Report Set Entry Values to and from Programs 9-40
Report Set Entry Properties Available for Passing 9-41
Passing Values from a Report Set Entry to a Program Instance 9-41
Passing Values from a Program Instance to the Report Set for Post-Processing 9-42
Installing Report Sets 9-44
Installing the Report Set as a Whole with All Programs Checked In 9-45
Installing the Report Set as a Whole with Some Programs Checked Out 9-45
Installing a Single Program Instance in the Report Set 9-46
Validating Report Set Definitions and Outputs 9-46
Output Reuse 9-47
Report Set Entry Properties and Reuse 9-48
Program Output Validation Flag 9-48
Report Set Validation Status 9-49
Summary Output Validation Status 9-50
Output-Oriented Validation 9-52
Definition-Oriented Validation 9-53
Changing Validation Status 9-54
Changing the Validation Status of an Output 9-54
Changing the Validation Status of a Report Set Entry 9-54
Changing the Validation Status of a Program Instance in a Report Set 9-55
About Report Set Planned Outputs 9-55
Modifying Report Sets 9-55
Modifying Report Set Instance Properties 9-56
Modifying Report Set Definition Properties 9-56
Modifying the Report Set Structure 9-57
Modifying Report Set Entries 9-57
Modifying Programs 9-57

10 Defining Workflows

About Workflows 10-1
- Workflow Components 10-2
- Workflow Rules 10-3
- Execution 10-3
- Reports on Workflow Definitions and Instances 10-3

Creating a Workflow 10-3
- Creating a New Workflow Definition and Instance 10-4
- Creating an Instance of an Existing Workflow 10-4

Using the Workflow Properties Screen 10-5
- Instance Properties 10-5
- Definition Properties 10-6
- Buttons 10-6

Adding Executables 10-6
- About Executables as Workflow Activities 10-7
- Adding an Executable to a Workflow 10-7
- Mapping Table Descriptors within a Workflow 10-8

Adding Workflow Structures 10-8
- About Workflow Structures 10-8
- Types of Workflow Structures 10-9
- Adding Structures 10-9

Defining Notifications 10-10
- About Notifications 10-10
  - Types of Notifications 10-10
- Using Approvals 10-11
- Creating a Notification 10-11
  - Creating a New Notification Definition and Instance 10-12
  - Creating an Instance of an Existing Notification 10-13
  - Specifying Notification Recipients 10-13
  - Defining a Link to a Planned Output 10-14
  - Writing Notification Messages 10-14
  - Defining Notification Parameters 10-15
- Modifying Notifications 10-15

Defining Transitions 10-16
- About Transitions 10-16
- Creating Transitions 10-16

Defining Workflow Parameters 10-17

Workflow Planned Outputs 10-17

Installing Workflow Instances 10-18
- Accessing the Log File 10-18

Modifying Workflows 10-19
Modifying Workflow Instance Properties 10-19
Modifying Workflow Definition Properties 10-20
Modifying Activities and Transitions 10-20
Modifying Parameters 10-20
Modifying Table Descriptor Mappings 10-21

11 Defining Business Areas for Visualizations

About Visualizations 11-1
Creating a Business Area 11-2
Creating a New Business Area Definition and Instance 11-3
Creating an Instance of an Existing Business Area Definition 11-4
Using the Business Area Properties Screen 11-4
Instance Properties 11-4
Definition Properties 11-5
Business Area Attributes 11-6
Buttons 11-6
Defining Table Descriptors 11-6
Defining Joins 11-7
Defining a Join at the Table Level 11-8
Defining a Join at the Column Level 11-9
Defining Business Area Hierarchies 11-10
Grouping Columns 11-10
Understanding Business Area Source Code 11-12
Setting Business Area Attributes and Parameters 11-12
Defining Oracle Business Intelligence Business Areas 11-12
About OBIEE Business Areas 11-12
Defining an OBIEE Business Area 11-13
Manually Deploying the Master RPD File 11-15
Starting the WebLogic Server 11-16
Visualizing Business Area Data using OBIEE Answers 11-16
OBIEE Security 11-17
End User Security 11-17
Developer Security 11-17
Installing and Setting Up Oracle Business Intelligence Administration Tool 11-17
Set Up Database Connectivity 11-18
Defining Generic Visualization Business Areas 11-18
Generic Visualization Business Area Instance Properties 11-19
Assigning Security Privileges to Business Area Data 11-20
Launching Visualizations 11-21

Creating a Visualization 11-21
Setting Data Currency and Blinding Values 11-21
Setting the Blind Break Value 11-21
Setting the Shared Snapshot Value 11-22
Installing Business Area Instances 11-22
Accessing the Log File 11-23
Modifying Business Areas 11-23
Modifying Business Area Instance Properties 11-24
Modifying Business Area Definition Properties 11-25
Modifying Table Descriptors 11-25
Modifying Joins 11-26
Modifying Business Area Hierarchies 11-26
Modifying Business Area Source Code 11-26

12 Using, Installing, and Cloning Work Areas

Using the Work Area Properties Screen 12-1
Work Area Properties 12-1
Viewing a Work Area's Installation History 12-3
Viewing a Work Area's Version History 12-4
Version Details 12-5
Changing a Work Area's Usage Intent 12-5
Object Instance Information 12-5
Object Instance Actions 12-8
Using the Icons in the Object's Row 12-8
Using the Drop-Down List 12-8
Adding Object Instances to a Work Area 12-10
Managing Table Instance Snapshot Labels in a Work Area 12-10
Querying for Snapshots 12-10
Table Instance Information Displayed 12-11
Adding, Removing, or Moving a Snapshot Label 12-11
Personalizing Your Work Area Properties Screen 12-12
Installing a Work Area and Its Objects 12-13
About Work Area Installation 12-14
Running a Work Area Installation 12-16
Installing Individual Objects 12-18
Viewing Installation Results 12-18
Failed Installations 12-19
Successful Installations 12-20
What Happens During a Work Area Installation 12-20
Schemas 12-20

Object Name Resolution 12-21
Work Area Objects Converted to Oracle LSH Schema and OWB Objects 12-21
Work Area Installation Phases and Statuses 12-22
Work Area Status 12-24
Cloning Work Areas for Testing and Production 12-25
Application Life Cycle 12-26
Cloning a Work Area 12-29

13 Execution and Data Handling

About Execution 13-1
Submitting Jobs for Execution 13-2
Submitting a Job from the Applications Tab 13-2
Data Processing Types 13-2
Processing Types Summary 13-3
Transactional Processing 13-4
Reload Processing 13-5
Staging Processing 13-6
Transactional High Throughput Processing 13-6
SQL*Loader Processing for SAS Programs 13-8
Using Tables as Pass-Through Views 13-8
Data Auditing, Snapshots and Refresh Groups 13-8
Data Auditing 13-8
Auditing in Transactional and Reload Processing 13-9
Auditing in Staging Processing 13-9
Data Snapshots 13-9
Refresh Groups 13-10
Processing Data Subsets 13-10
Forward Chaining 13-11
Backchaining 13-12
How Backchaining Works 13-13
Backchaining Rules 13-15
Backchaining Tips 13-15
Submit the Execution Setup 13-16
Recover from a Canceled Job 13-16
Supply Default Parameter Values 13-16
Do Not Include Workflows with Approval Requests 13-16
Expect Slower Performance 13-16
Managing Blinded Data 13-16
Loading Real and Dummy Data 13-17
SAS and Text Load Sets 13-18

Oracle Tables and Views 13-18
Oracle Clinical Randomization 13-19
Managing Blinding Along the Data Flow 13-19
Unblinding Table Instances 13-20
Using Message-Triggered Submission from External Systems 13-21
About Message-Triggered Submission 13-21
Setup Required 13-22
XML Message Requirements 13-22
Executable Specification 13-23
System Parameter Values 13-23
User ID 13-24
Request ID 13-25
Required XML Schema 13-25

14 System Reports

Running and Saving System Reports 14-1
Alphabetical Listing of Reports 14-2
Security Reports 14-2
Blinding Rights Report 14-3
Operations for a Role Report 14-3
User Group Assignments Report 14-3
Users in Group Report 14-4
Data Blinding Reports 14-4
Blind Breaks Report 14-4
Blinded Table Instances Audit Report 14-5
Blinded Table Instances Report 14-5
Unblinded Outputs Report 14-5
Container Reports 14-6
Application Area Library Report 14-6
Domain Library Report 14-6
Work Area - All Instances Report 14-7
Work Area Cloning Report 14-7
Work Area Details 14-7
Work Area Installation History Report 14-8
Installation Details 14-8
Work Area Version History Report 14-8
Version Details 14-8
Object Metadata Reports 14-9
Common Header Information 14-9
Header Information for Object Definitions 14-9

Header Information for Object Instances 14-10
All Instances of a Definition Report 14-10
Data Mart Report 14-10
Data Mart Instance Report 14-11
Load Set Report 14-12
Load Set Instance Report 14-12
Object Validation Report 14-13
Object Version History Report 14-13
Program Report 14-13
Program Instance Report 14-14
Report Set Report 14-15
Report Set Entries 14-15
Program Instance Under Report Set Entry (Full Title) 14-16
Report Set Instance Report 14-16
Report Set Entries 14-17
Program Instance Under Report Set Entry (Full Title) 14-17
Table Report 14-17
Table Instance Report 14-18
Workflow Report 14-18
Workflow Instance Report 14-19

A Installation Requirements for Each Object Type

Installable Instances and their Definitions A-1
Component Object Types A-3
Organizational Objects A-4

B Object Ownership

Domains B-1
Application Areas B-5
Work Areas B-6

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Preface

This preface contains the following sections:

- Documentation accessibility
- Related resources

Documentation accessibility

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

Related resources

All documentation and other supporting materials are available on the Oracle Help Center.

1

Getting Started

For an overview of Oracle Life Sciences Data Hub, see "Overview" in the *Oracle Life Sciences Data Hub Implementation Guide*.

See also Common Development Tasks.

- About Application Development
  The Oracle Life Sciences Data Hub allows you to load data from different external systems (and from different trials on the same system), merge data and produce reports for interim analysis and clinical trial reporting, all in compliance with industry regulations.
- Defining Objects
  To develop an application in Oracle Life Sciences Data Hub, you **define objects**. For example, in Oracle Life Sciences Hub a Program is a defined object. In addition to writing program source code in a development environment, you must define a Program and each of its components (Parameters, Source Code, and other components) as objects in the Oracle Life Sciences Hub user interface.
- Developing a Business Application
  An *Applications* application may be as complex as storing all the data and performing all the analysis required for a study, or as simple as running a single set of reports.
- Developing Standard Definitions and Modular Applications
  You can reduce the amount of time required to develop and validate applications by creating standard object definitions and reusing them as much as possible.
- Ensuring Data Currency
  Oracle Life Sciences Data Hub provides two different ways to ensure that data in an Oracle Life Sciences Hub Table or report reflects the most current source data: Workflows and backchaining execution.

About Application Development

The Oracle Life Sciences Data Hub allows you to load data from different external systems (and from different trials on the same system), merge data and produce reports for interim analysis and clinical trial reporting, all in compliance with industry regulations.

The Oracle Life Sciences Hub allows you to load data from different external systems (and from different trials on the same system), merge data and produce reports for interim analysis and clinical trial reporting, all in compliance with industry regulations.

To accomplish any of these goals you must develop an *application*. An application typically loads data from an external system or reads Oracle Life Sciences Hub data, merges and/or transforms the data, and produces one or more of the following outputs:

- **Reports**. An application may generate simple reports—figures, listings, and tabulations—on data in any standard format such as TXT or EPS.

- **Report Sets**. An application may create a complex single- or multi-volume set of reports with a single table of contents. With Oracle XML Publisher you can create custom templates and use them to produce PDF outputs.
- **Data Marts**. An application may generate large data files in any of the following formats: text, Oracle Export, SAS X-port, SAS C-port, or SAS data sets. (The SAS formats are available only if you have SAS).
- **Data Visualizations**. An application may make data available so that users can create ad hoc data queries in a tabular or graphical display using Oracle Business Intelligence Enterprise Edition.

Integrated Development Environments (IDEs): To transform data or produce reports, including the reports in Report Sets, you can use one of the several development environments, including SAS, SQL*Plus, SQL Developer, Oracle Reports, Oracle Business Intelligence Publisher (BIP) or any other system for which your company has built or bought an Oracle Life Sciences Hub adapter (products not produced by Oracle are not included with Oracle Life Sciences Hub and all Oracle products may not be included either; check with Oracle Sales for information). In each environment, you write a program or create a report as usual in that environment. You then store the source code in Oracle Life Sciences Hub under version control. When you run the program in Oracle Life Sciences Hub, the system sends the job to the appropriate engine for execution and stores the results in Oracle Life Sciences Hub with the security you specify.

For example, if you have SAS installed on your personal computer you can launch SAS directly from Oracle LSH and view Oracle Life Sciences Hub data from SAS as you develop your SAS program.

Using Existing Programs: If you have already developed SAS, PL/SQL, or Oracle Reports programs that you want to continue to use, you can upload them to Oracle Life Sciences Hub.

Using Existing Data Sets, Tables, and Data Text Files: You can load SAS data sets, Oracle tables, or text data files into Oracle Life Sciences Hub. Oracle Life Sciences Hub can read the structure of data sets and tables in an external system and automatically create Oracle Life Sciences Hub Tables with the same structure: Columns with the same data types, length, and name as the external table columns or data set variables. (Every Oracle Life Sciences Hub Table has all the required attributes of a SAS data set and an Oracle table.)

You can load data into Oracle Life Sciences Hub using text files in either fixed or delimited format. However in this case you must manually create the target Oracle Life Sciences Hub Tables before loading the data into them.

If you use Oracle Clinical, you can import all your Oracle Clinical Global Library questions, question groups, and discrete value groups, which Oracle Life Sciences Hub converts automatically to Oracle Life Sciences Hub Tables and other data structures.

Defining Objects

To develop an application in Oracle Life Sciences Data Hub, you **define objects**. For example, in Oracle Life Sciences Hub a Program is a defined object. In addition to writing program source code in a development environment, you must define a Program and each of its components (Parameters, Source Code, and other components) as objects in the Oracle Life Sciences Hub user interface.

Oracle Life Sciences Hub stores the objects you define as metadata in the Oracle LSH database under version control. This approach has the following advantages:

- **Validation**. All defined objects have a validation status that allows you to track whether an object you are using has been fully tested and validated, for regulatory compliance. Changes to validated objects are tracked so that they can be revalidated. Your company must develop validation standards. You can validate legacy programs that you have migrated into Oracle Life Sciences Hub, as well as Programs and other defined objects that you create in Oracle LSH. See Validating Objects and Outputs.
- **Reuse**. You can reuse object definitions, which ultimately saves work, minimizes the validation required, and promotes consistency. You can design and create a set of modular, small-scope Programs to promote reuse and validation (see Object Definitions and Instances, and their Containers and Developing Standard Definitions and Modular Applications).
- **Security**. The Oracle Life Sciences Hub security system allows you to control access to, and operations on, defined objects and their outputs (primarily reports and report sets); see Applying Security to Objects and Outputs .
- **Classification**. Your company can develop a classification system to label defined objects and their outputs and to create a customized user interface for outputs. Users can use these classifications to search for objects and outputs (see Classifying Objects and Outputs).
- Predefined Object Types
  Oracle Life Sciences Data Hub includes a set of predefined object types.
- Object Definitions and Instances, and their Containers
  To promote the reuse of object definitions, Oracle Life Sciences Data Hub allows you to create multiple instances of the same object definition.

Predefined Object Types

Oracle Life Sciences Data Hub includes a set of predefined object types.

Each object type is specialized for a particular data handling function. There is a detailed chapter on each type of object in this book. For an overview of how all the object types fit together, see "Overview" in the *Oracle Life Sciences Data Hub Implementation Guide*. You can define the following types of objects:

- **Tables**. Oracle Life Sciences Hub Tables have characteristics of both Oracle tables and SAS data sets, and can hold data from both SAS and Oracle source systems at the same time. See Defining Tables.
- **Programs**. Programs transform and/or report data using source code of a single technology type: SAS, PL/SQL, or Oracle Reports. See Defining Programs.
- **Load Sets**. Load Sets load data into Oracle LSH Tables from external systems that are integrated with Oracle Life Sciences Hub by an adapter or set of adapters. See Defining Load Sets.
- **Report Sets**. Report Sets allow you to define the hierarchical structure of a set of reports, with each chapter and subchapter represented by an Entry in the Report Set. Each Report Set Entry can contain a Program that produces the content of that Chapter (for example, a SAS program that generates a summary table). The Report Set Entry can also contain text (called a *narrative*) that is included in the output.

When you submit a Report Set for execution, the system executes all the Programs on current data (or a data snapshot, if you have specified a snapshot) and creates a table of contents. If you use Oracle XML Publisher, the Report Set collates the outputs into one or more PDF documents based on custom templates. See Defining Report Sets.

- **Data Marts**. Data Marts allow you to export data stored in Oracle Life Sciences Hub to a file. These files can be in a variety of different formats depending upon where you would like to use the Data Mart's output, including Oracle Export, SAS data sets, SAS transport, or text. See Defining Data Marts.
- **Workflows**. You can use a Workflow to string together other Oracle Life Sciences Hub executables—Load Sets, Programs, Report Sets, or Data Marts—so that they execute as part of a single process in a predefined order. For example, if you have one SAS Program that converts raw data sets into merged data sets and another SAS Program that produces a summary report on the merged data, you can use a Workflow to execute those Programs in order.
  A Workflow can contain conditional branches to handle the success or failure of each program or other activity. A Workflow can also generate Notifications—customized email messages and Oracle Life Sciences Hub Home Page messages to users—at specified points in the Workflow. There are two types of Notifications: those that request an action from the recipient (Approvals) and those that simply provide information (FYI). See Defining Workflows.
- **Business Areas**. Business Areas provide a view onto Oracle Life Sciences Hub data for Oracle Business Intelligence Enterprise Edition for the purpose of allowing nonprogramming personnel to create ad hoc data visualizations. See Defining Business Areas for Visualizations.

Each of these objects contains subcomponent objects that you must also define. Details are included in the chapter on each type of object. Parameters and Variables have their own chapter, Defining Variables and Parameters.

Object Definitions and Instances, and their Containers

To promote the reuse of object definitions, Oracle Life Sciences Data Hub allows you to create multiple instances of the same object definition.

The object instance is also a defined object, but it consists primarily of a pointer to the definition.

For example, if you have a standard demography table, you can create instances of the Table definition for use in many different studies. You may also have a standard Program that reads from the Demography table to create a report. You can create instances of the Program definition for use in different studies as well.

- Organizational Objects
- Object Definitions and Instances
  In order to use an object you must create both the object definition and an instance of it in a Work Area, and install the instance.

Organizational Objects

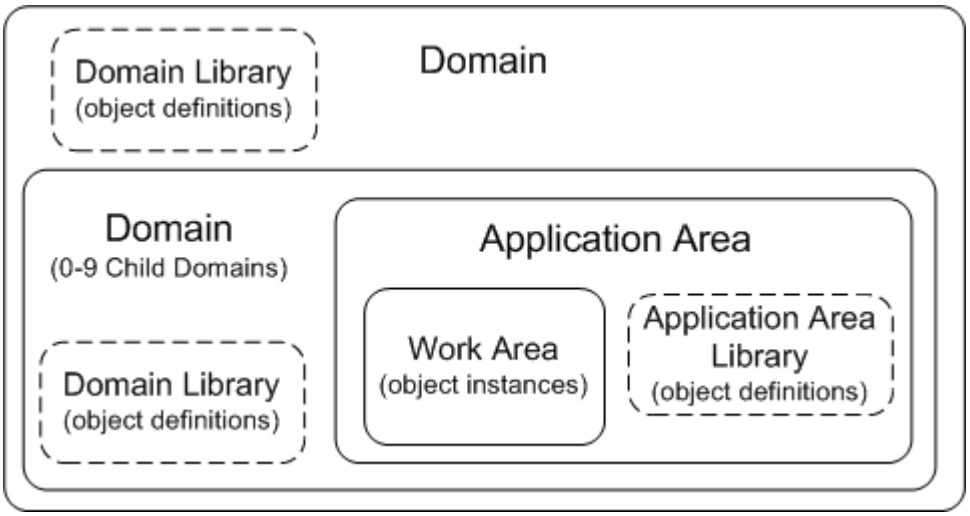
Oracle Life Sciences Data Hub has three types of container objects that hold object definitions and instances. These containers are nested, or contained within one another, as shown in Figure 1-1 and described as follows:

- **Domains**. Domains contain Application Areas and object definitions that have been explicitly moved into the Domain.
  In addition, Domains can contain child Domains that themselves contain child Domains, up to nine (9) levels, depending on the value of the Domain Nest Value profile setting for your Oracle LSH implementation. A Domain can contain any number of Domains at a single level. For example, if the Domain Nest Value profile is set to one (1), a top-level Domain can contain any number of child Domains, but those child Domains cannot contain any child Domains of their own.

  You can use child and grandchild Domains to more finely organize object definitions and applications.

  In general, Domain security should be set up so that only a few people can create, modify, or move object definitions in Domains. Only object definitions that have been thoroughly tested and approved for reuse should be moved into Domains.
- **Application Areas**. An Application Area contains a library of definitions developed specifically for a particular application and the Work Areas necessary to develop, test, and put the application into production.
- **Work Areas**. A Work Area contains instances of all the object definitions required for a particular application.

Figure 1-1 Oracle Life Sciences Hub Organizational Structure



For information on planning an organizational structure of Domains, Libraries, Application Areas and Work Areas for your company, see "Designing an Organizational Structure" in the *Oracle Life Sciences Data Hub Implementation Guide*.

Object Definitions and Instances

In order to use an object you must create both the object definition and an instance of it in a Work Area, and install the instance.

If you work in a Work Area, the system automatically creates both the definition and the instance at the same time.

The *primary* objects you define—Tables, Programs, Load Sets, Report Sets, Data Marts, and Workflows—contain secondary objects. For example, a Table contains

Columns and Constraints, and a Program contains Source Code, Parameters, Table Descriptors, Planned Outputs, and Execution Templates.

Object instances also contain objects, notably Execution Setups and Mappings. See Object Ownership for further information.

Modifying Object Definitions: To modify an object definition, you must first check it out. Checking it out creates a new version of the definition—the one you are working on. No one else can check out the definition until you either check it in or undo the checkout. Existing versions of the object continue to function as before while you have the definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.

The exception to this rule is Report Sets. Report Set definitions have a shared checkout system, so that after one user checks out a Report Set, anyone with the necessary security privileges on the Report Set can modify it. The system locks the sections people are working on to prevent conflicting modifications.

If you modify an object contained in the definition, the system implicitly checks out any secondary object you modify and implicitly checks it in again when you apply your changes. For example, if you check out a Program, modify and upload your Source Code, the system implicitly checks the Source Code out and then in again when you apply your changes.

Modifying Object Instances: Anyone with Modify privileges on an object instance can modify it without checking it out. However, when you click the **Update** button to begin modifying the object instance, the system creates a lock on the object so that no one else can modify it at the same time. The system also implicitly checks out the underlying object definition and increments its version number. When you click **Apply**, the system releases the lock and implicitly checks in the definition.

Developing a Business Application

An *Applications* application may be as complex as storing all the data and performing all the analysis required for a study, or as simple as running a single set of reports.

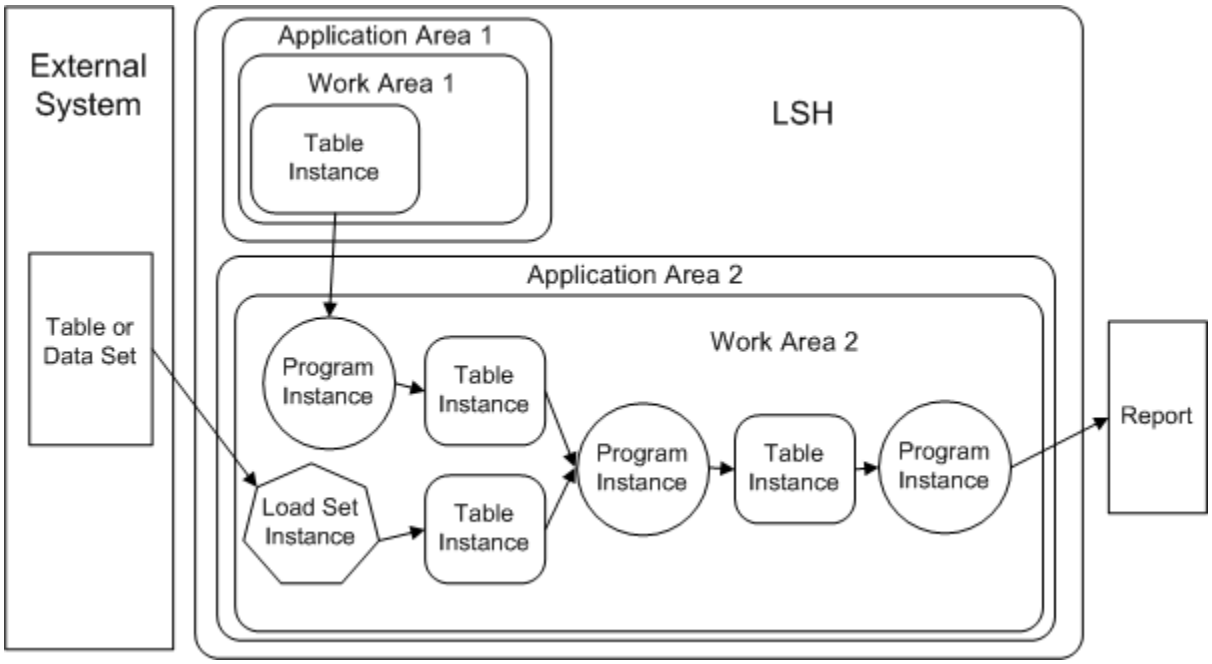
An application typically includes the following general steps, each of which requires a particular defined object type:

1. Make data available for processing by doing one or both of the following:
   - Load data from one or more external systems using Load Sets.
   - Use Programs to read data already stored in Oracle Life Sciences Data Hub.
2. Use Programs to transform the data in one or more steps.
3. Report and/or transport data out of Oracle Life Sciences Hub. Use Programs and Report Sets to generate reports. Use Data Marts to write data to files for transport out of Oracle Life Sciences Hub.

Use Tables to store data at each stage of processing in Oracle Life Sciences Hub.

You can use a Workflow to combine some or all of these steps into a single executable process as shown in Figure 1-2. See Predefined Object Types for a description of each of these object types.

Figure 1-2 Example of Data Flow in a Work Area



- Working in a Work Area
  As an Oracle Life Sciences Data Hub application developer, or Definer, you work in a Work Area, creating instances of all the object definitions—Tables, Programs, Load Sets, Report Sets, Workflows, Data Marts, and/or Business Areas—required by the application.
- Mapping Executables to Tables
  To promote the reusability of Programs and other executable object definitions, Oracle Life Sciences Data Hub requires adding a subcomponent called a Table Descriptor to definitions of executable objects for each Table instance the executable reads from or writes to.
- Example of a Business Application
  You have both a SAS and an Oracle system integrated with Oracle Life Sciences Data Hub, with demography and adverse events data for Study A in the SAS system and demography and adverse events data for Study B in the Oracle system.

Working in a Work Area

As an Oracle Life Sciences Data Hub application developer, or Definer, you work in a Work Area, creating instances of all the object definitions—Tables, Programs, Load Sets, Report Sets, Workflows, Data Marts, and/or Business Areas—required by the application.

Work Areas contain only object instances, but you can see the properties of the underlying object definition in the Work Area as well. In general, the instance properties are displayed in the upper part of the Properties screen for the object, while the definition properties are displayed in the lower part of the screen.

Creating New Object Definitions: When you create an object instance in a Work Area, you have the choice between creating an instance of an existing definition or creating a new definition and instance at the same time. When you create a new definition and instance at the same time, the system simultaneously creates the object definition in the Application Area that contains the Work Area, and an instance of that definition in the Work Area. You work on the definition through the instance in the Work Area.

Follow your company's standard operating procedures (SOPs) for testing and validating each object you create (see Validating Objects and Outputs).

Creating Instances of Existing Definitions: When your company begins to use Oracle Life Sciences Hub, you will need to create many new object definitions. However, as time goes by your company will develop libraries of tested and validated object definitions. Your company may choose to move validated definitions that are suitable for reuse in other applications into Domain libraries.

If you create an instance of an existing validated object and do not modify it, you do not need to revalidate the definition. If you need to modify the definition because it is flawed or needs to be updated, you can do so. You can then update some or all instances of the definition. If you need to modify a definition because you need different functionality, you can copy the definition and modify the copy. See Reusing Existing Definitions for further information.

Installing Work Areas: You must *install* a Work Area and all its object instances to the database in order to run executable objects (Programs, Load Sets, Report Sets, Workflows, and Data Marts) or write data into Tables. See Installing a Work Area and Its Objects for further information.

Mapping Executables to Tables

To promote the reusability of Programs and other executable object definitions, Oracle Life Sciences Data Hub requires adding a subcomponent called a Table Descriptor to definitions of executable objects for each Table instance the executable reads from or writes to.

A Table Descriptor is very similar to a Table instance in that it consists of a pointer to a Table definition. The Table Descriptor acts as the interface between the executable and its source and target Table instances. You must map the Table Descriptor to the source or target Table to enable the executable to read from or write to the Table instance.

This extra definitional layer facilitates the reuse of executable definitions. The executable source code refers to source and target tables using the names of the Table Descriptors included in the executable definition. Those Table Descriptors can be mapped to Table instances with the same name and structure or a different name and, if Column data types and lengths are compatible, a different structure.

For example, if the demographic table is called DEMOG in one study and DEMO in another, you can use the same program to generate a demography report in both studies, if the data types and lengths of the corresponding Columns are compatible.

Also, if a Program reads from some but not all Columns of a table, you can create a Table Descriptor with only the necessary Columns and map it only to the required Columns in the Table instance. In that case, it does not matter if the unmapped Columns are incompatible.

In many cases, Oracle Life Sciences Hub can automatically map Table Descriptors to Table instances. If necessary, you can map manually. See Mapping Table Descriptors to Table Instances for further information.

Example of a Business Application

You have both a SAS and an Oracle system integrated with Oracle Life Sciences Data Hub, with demography and adverse events data for Study A in the SAS system and demography and adverse events data for Study B in the Oracle system.

You want to merge and compare the data from the two studies.

To accomplish this you can do the following (see Figure 1-3):

1. Define Load Set definitions and instances: you need one Oracle-type Load Set to load both Oracle tables. You need one or two Load Sets for the SAS data sets. If you use a SAS transport file containing both data sets you can load both data sets with a single Load Set. If you load the data set files directly you need two Load Sets, one for each. During Load Set definition, you specify the external tables and data sets to load. The system does the following:
   - Oracle Life Sciences Hub creates target Table Descriptors with the same metadata structure as the source tables and data sets.
   - When you invoke the Table Instances from Existing Table Descriptors job, Oracle Life Sciences Hub creates a matching Table definition and instance for each source table and data set—a total of four Table definitions and instances for Study A Adverse Events, Study A Demography, Study B Adverse Events, and Study B Demography.
   - When you invoke the Automatic Mapping By Name job, Oracle Life Sciences Hub maps each Table Descriptor to its corresponding Table instance.
2. Create instances of two previously defined standard Oracle Life Sciences Hub Tables, Demography and Adverse Events, to hold the merged data.
3. Define a Program definition and instance to merge the data from the SAS Demography data set and the Oracle Demography table into your standard Oracle Life Sciences Hub Demography Table. Name the Program "Merge Oracle and SAS Demog."

   In the Program definition, create two source Table Descriptors based on the same Table definition as the Load Set target Table instance. Invoke the Automatic Mapping By Name job to map the Table Descriptors to the Table instances with the same name.

   In the Program definition, also create one target Table Descriptor based on your standard Oracle Life Sciences Hub Table called Demography. In addition, create an instance of the standard Demography Table to receive the data generated by the Program. Invoke automatic mapping by name.
4. Define a Program definition and instance to merge the data from the SAS Adverse Events data set and the Oracle Adverse Events table into your standard Oracle Life Sciences Hub Adverse Events Table. Name the Program "Merge Oracle and SAS AE."

   In the Program definition, create two source Table Descriptors based on the same Table definition as the Load Set target Table instance. Invoke automatic mapping by name.

In the Program definition, create one target Table Descriptor based on your standard Oracle Life Sciences Hub Table called Adverse Events. In addition, create an instance of the standard Adverse Events Table to receive the data generated by the Program. Invoke automatic mapping by name.

5. Define a Program definition and instance to read data from both the Demography and Adverse Events Table instances and produce a report comparing adverse events in Study A to those of Study B. Name the Program "Compare Adverse Events in Two Studies."

   In the Program definition, create two source Table Descriptors based on your standard Oracle Life Sciences Hub Tables, one for Adverse Events and one for Demography. Invoke automatic mapping by name.

   The Program definition does not need any target Table Descriptors because it does not write data to any Table instances. It does need a Planned Output to create the actual report.

6. Define another Program definition and instance to read data from the Demography Table instance and compare patient information from Study A to patient information from Study B. Name the Program "Compare Demography from Two Studies."

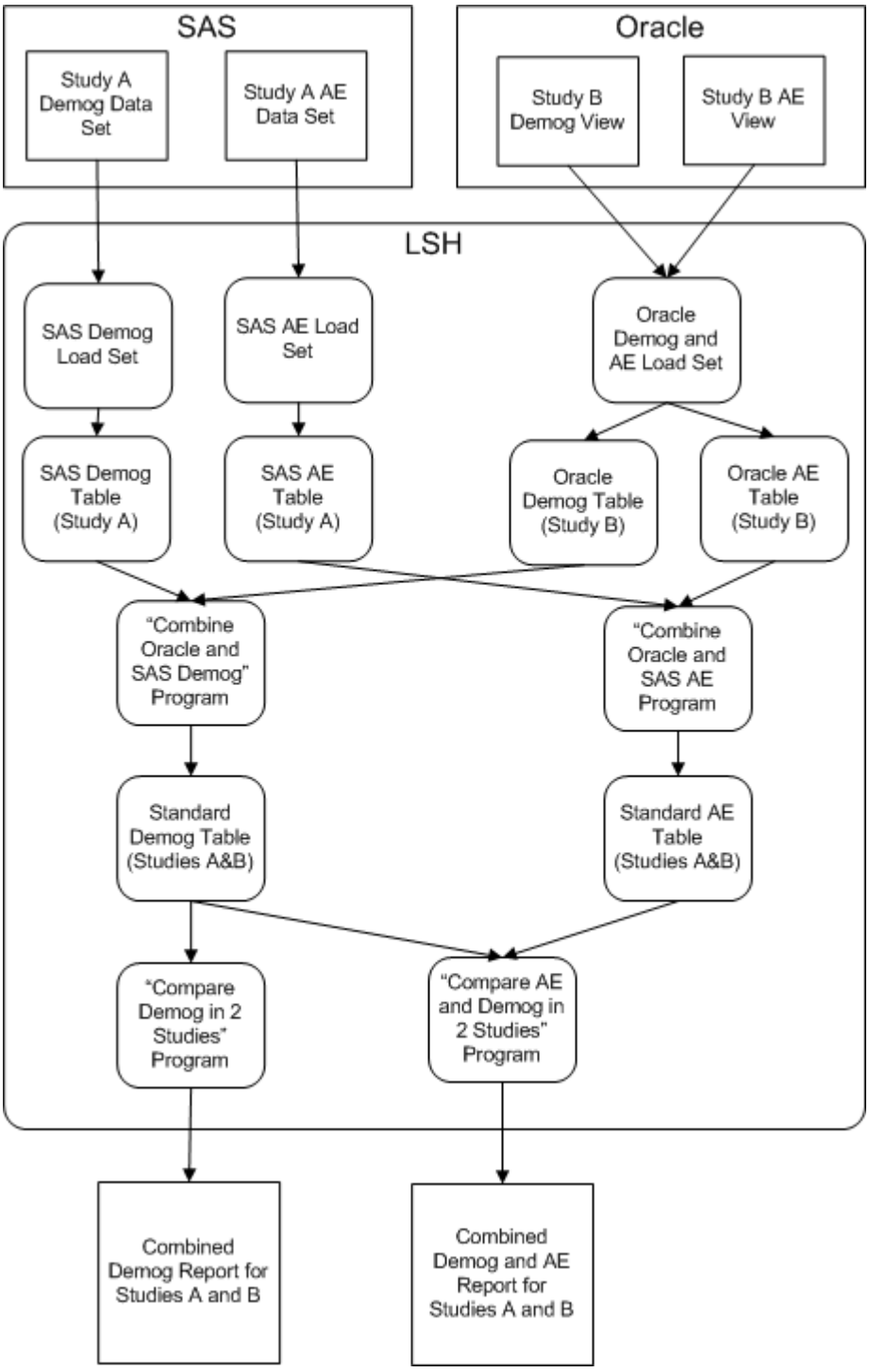
   In the Program definition, create one source Table Descriptor based on the standard Demography Table. Invoke automatic mapping by name.

   The Program definition does not have any target Table Descriptors because it does not write data to any Table instances. It does have a Planned Output to create the actual report.

You can update data periodically by rerunning the Load Set instances. You can then rerun the other Program instances in turn to update the report data.

If you would like to load the data, run the Programs, and generate the reports in a single process, you can create a Workflow definition containing all the Load Set and Program instances and defining the order in which they must be executed. You can also include an email notification as part of the workflow to be automatically sent to a group of people to alert them when the report is generated.

Figure 1-3 Business Application Example



Good Practice in Application Design: Although it is possible to write one Program instead of four in the above example to accomplish the same result, it is good practice to divide the operations needed into discrete parts and define smaller Programs to perform each part. These smaller-scope Programs are more generic and therefore more reusable than one big Program. See Developing Standard Definitions and Modular Applications.

Developing Standard Definitions and Modular Applications

You can reduce the amount of time required to develop and validate applications by creating standard object definitions and reusing them as much as possible.

To encourage Definers to reuse standard definitions, make it clear which definitions are standard by storing them in a library created especially for that purpose and/or classifying them as Standard, for example.

Following are a few strategies for developing fstandard object definitions:

- **Develop standard Table definitions**. If you have a standard set of Tables, with standard data types and lengths for corresponding Columns in different Tables, then you can easily reuse the same Table definition at different points in the data flow, and write standard Programs to read from and write to the standard Tables. You can develop your own company standards or use external standards such as the CDISC data model.

  The standard data model you use must be compatible with the data model of the source system(s) you use. For example, if the Patient ID column or variable has a length of 8 in the source system, the Patient ID Column in a standard Oracle Life Sciences Data Hub Table must have a length of 8 or more.

- **Define more Programs with a smaller scope** instead of fewer Programs with a larger scope. You can then use the smaller-scope Programs as modules in multiple applications.
  For example, rather than define a single Program to merge data from different studies and then analyze the data, define two separate Programs, one to merge data and the other to analyze data. Both Programs are reusable in more situations. You can reuse the combining Program before different analytical Programs, and you can reuse the analytical Program, with minor modifications, on data from either a single study or multiple studies.

  You can include a series of Programs in a single Workflow, so that you can run all the Programs in a single process, so that there is no time lost by running separate Programs; the successful execution of one Program triggers the next in the series, and some Programs can run in parallel.

- **Use Parameters**. Make executable object definitions more reusable by defining Parameters to contain information that may clearly change from one use of the definition to another.
  For example, if you want to use the same Program to generate a Demography Report in several different studies, create a Parameter to contain the study name. You can either make the Study Parameter enterable by the user who runs the report, or, in the Execution Setup for each instance, bind the Study Parameter to the value appropriate for each study.

- **Create Parameter Sets based on standard Table definitions**. Both Oracle Life Sciences Hub Parameter and Column definitions are based on a data structure called an Oracle Life Sciences Hub Variable, which determines their data type and length. You can create Parameter Sets containing Parameters based on the same Variables on which a Table's Columns are based, and give the Parameter Set the same name as the Table so that, when you are defining Program Parameters, you can easily find Parameters with the same data type and length as the corresponding Table Column. This approach has two advantages:

  – It promotes data type and length consistency along the data flow.

– It makes automatic Parameter value propagation in Workflows and Report Sets easier to set up.

Ensuring Data Currency

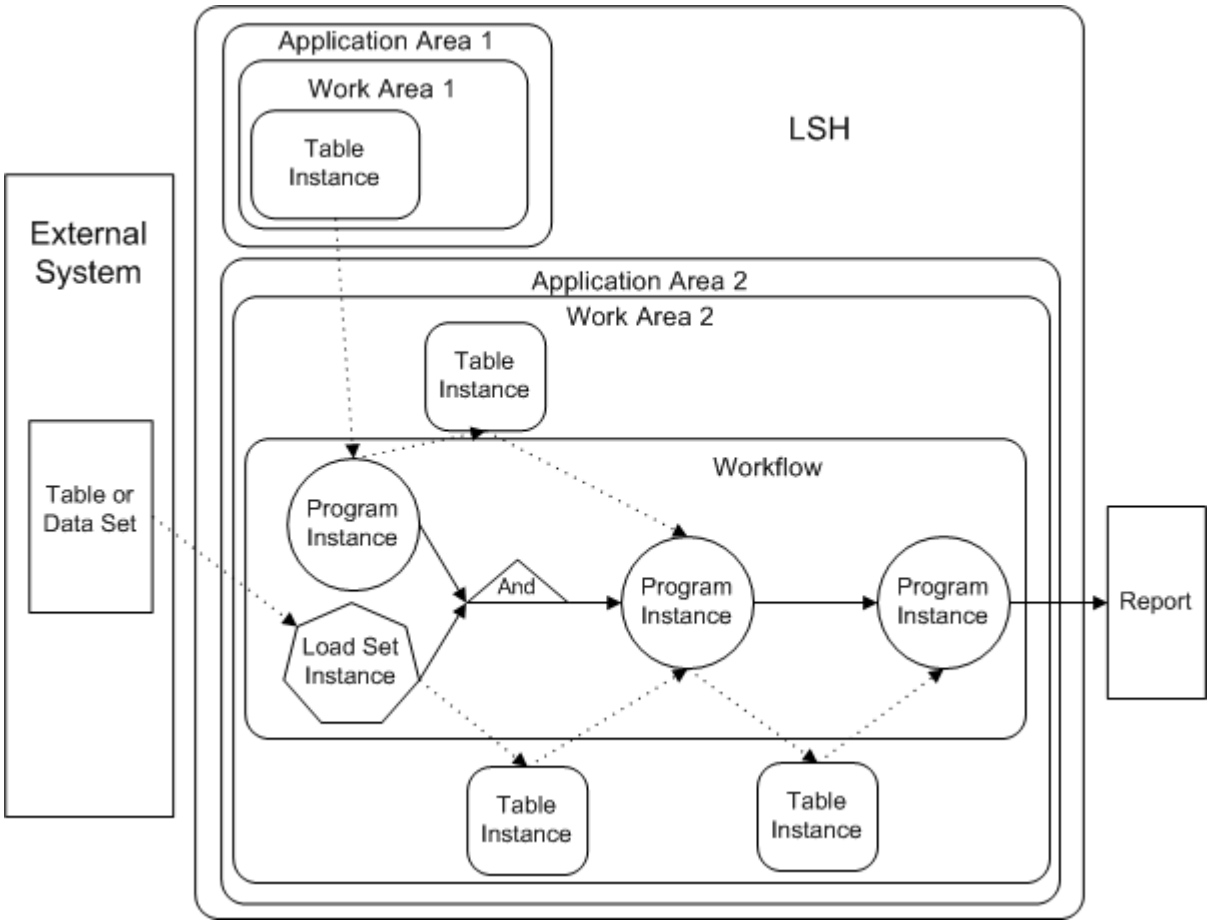
Oracle Life Sciences Data Hub provides two different ways to ensure that data in an Oracle Life Sciences Hub Table or report reflects the most current source data: Workflows and backchaining execution.

A Workflow pushes the most current data forward, while a backchain goes backward along the data flow to find more recent data and pull it into the current job. In both cases, you must explicitly define the entire data flow from the source of the most current data to the job to be run on the most current data.

A Workflow is a defined object. Backchaining is a type of execution that you must specify as part of the Execution Setup for each executable along the data flow.

To ensure that the most current data is displayed in the report in Figure 1-4 below, you can use either a Workflow or a Backchain; see below.

Figure 1-4 Example of a Workflow



Workflow: To use a Workflow to ensure data currency, add all the executables — Load Sets and Programs—in Work Area 2 in Figure 1-4 to a Workflow in the same

Work Area and define the order in which they should be executed. The Load Sets and Programs write to and read from the Tables as they do without a Workflow, but the Tables are not part of the Workflow. When you execute the Workflow, the Load Set and the Program that pulls data in from Work Area 1 can both run at the same time. The next Program waits until both complete successfully, and then it runs. When it completes successfully, the final Program runs, generating the report.

You can schedule a Workflow to run at regular intervals. If the external system is Oracle Clinical, or if you have set up XML messaging in a different external system, you can schedule a Workflow to run when triggered by the successful completion of a job in the external system, such as batch validation in Oracle Clinical.

See Defining Workflows for further information.

Backchain: You must define at least one Execution Setup for each executable object definition in any circumstances. The Execution Setup serves as the basis for the submittal form that users need to run the executable.

To use backchaining to ensure data currency in this example, define an Execution Setup especially for backchaining for each of the Programs and the Load Set that feed data into the final Program. When a user runs the final Program (the one that generates the report) with the Data Currency system parameter set to Most Current Available, the system checks the Programs that feed data into the Report Program's source Table instances to see if those Program instances have Execution Setups with backchaining enabled.

If so, the system continues to check upstream for backchain Execution Setups in the Load Sets or Programs that feed data into those source Table instances. The system then compares the currency of the source and target data for each Program or Load Set that has a backchain Execution Setup, and executes the Program or Load Set if its source data is more current than its target data.

The system then runs each Program downstream in the data flow until the report Program's source Table instances have the most current possible data, and then executes the program that generates the report.

See Backchaining for further information.

2

Creating Container Objects

The *Oracle Life Sciences Data Hub User's Guide* includes information on the following related topics: Standard buttons and icons, and searching.

The Applications tab of the Oracle Life Sciences Data Hub is where you do almost all your work as an Oracle Life Sciences Hub application developer. Your company can design its own organizational structure consisting of nested Domains and Application Areas. Domains and Application Areas can contain libraries of object definitions. Application Areas also contain Work Areas, where you create instances of object definitions whose purpose is to load, manipulate, report, and view data in Oracle Life Sciences Hub. See Getting Started for further information.

- Selecting a Domain
  While you are working on applications you can see one top-level Domain at a time.
- Navigating in the Applications Tab
  This section includes the following topics:
- Creating Domains, Application Areas, Work Areas, and Objects
  Before creating Domains, Application Areas, and Work Areas, you should carefully design a set of these structures.
- Modifying Domains, Application Areas, and Work Areas
  Domains, Application Areas, and Work Areas should be part of a carefully designed organizational structure; see "Designing an Organizational Structure" in the *Oracle Life Sciences Data Hub Implementation Guide*.
- Duplicating, Removing, Moving and Promoting Domains
  You must have special privileges to duplicate or remove a Domain.
- Managing Object Definitions
  You can work on Oracle LSH object definitions within a Domain or Application Area from the Maintain Domain or Maintain Application Area screen.

Selecting a Domain

While you are working on applications you can see one top-level Domain at a time.

You can select the Domain you see in two ways:

- Set a Default Domain in Preferences
- Select a Domain on the Applications Screen

Set a Default Domain in Preferences

You can select a default Domain through the Preferences link at the top of any screen. Whenever you log in to Oracle Life Sciences Data Hub and click the Applications tab, you will see the Domain you select in Preferences. For instructions, see "Setting Preferences" in the *Oracle Life Sciences Data Hub User's Guide*.

If you do not select a Domain in Preferences, each time you log in to Oracle Life Sciences Hub you see the Domain that is before all other Domains in alphabetical/ numerical order. You can then select a different Domain using the instructions below.

Select a Domain on the Applications Screen

You can change Domains at any time in the main Applications screen:

1. Click the **Applications** tab.
2. In the **Select Domain** field do one of the following:
3. Enter the text of the second step here.
    a. **Enter part of the name and use a wildcard**; for example, if you enter %123% the system retrieves all Domains that have 123 in sequence in any position in their name.
    b. **Enter part of the Domain name and press Tab**. Do not enter any special characters.
4. The system opens the Search and Select window. Click the icon in the **Quick Select** column for the Domain you want. The system returns to the Applications screen and displays the Domain name in the **Select Domain** field.
5. Click **Go**.

Navigating in the Applications Tab

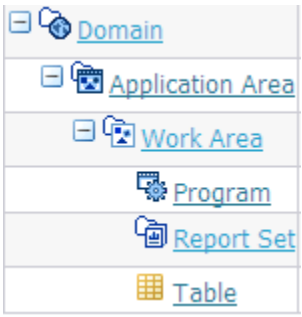
This section includes the following topics:

- Browsing the Main Applications Screen
- Navigating Using Breadcrumbs
- Using the Actions Drop-Down

Browsing the Main Applications Screen

In the main Applications screen you can see all the object definitions and instances in a single Domain. To change Domains, see Select a Domain on the Applications Screen.

Objects are displayed in a tree structure with expandable nodes, or branches as shown in the following image. Oracle Life Sciences Data Hub displays all objects within their containing object when you expand the node of the containing object. See Object Ownership for information on object containing relationships in Oracle Life Sciences Hub.



Navigating by Object Ownership

Within a Domain you see all the Domain's Application Areas and child Domains. Within each Application Area you see all its Work Areas. Within each Work Area you see all the objects it contains—instances of Tables, Programs, Load Sets, Data Marts, Workflows, Report Sets, and Business Areas. Within Report Sets you see the hierarchical Report Set Entry structure that corresponds to the Report Set's table of contents.

Viewing Object Definitions

To see the definitions contained in a Domain or Application Area library, do the following:

- Click the icon in the Manage Definitions column next to the Domain or Application Area name.
  Click the icon in the Manage Definitions column next to the Domain or Application Area name.

  Alternatively, you can click the hyperlink of the Domain or Application Area name and then click the **Manage Definitions** button from the Domain or Application Area's Properties screen.
- Click the icon in the Manage Definitions column next to the Domain or Application Area name.

  Alternatively, you can click the hyperlink of the Domain or Application Area name and then click the **Manage Definitions** button from the Domain or Application Area's Properties screen.
- All object types are listed alphabetically. Click **Expand All** to view all object definitions for all object types.
  Alternatively, you can expand the plus icon (+) next to an object type to view all the objects of that type in that Domain or Application Area.

  If you want to specifically see only one type of object, then you can click in the Focus column next to that object type. This is particularly useful if you have many objects of one type and want to minimize scrolling up and down the screen.
- Click the definition name whose details you want to see. The following image shows the definition icon.

> **Note:**
>
> When you work on an object definition through an instance of it in a Work Area, you can see almost all the properties of the definition. However, to see all of a definition's properties, including its definition, security assignments and classifications, you must either navigate to it in its Domain or Application Area, or jump to it through its link on the object instance screen.

Using Hyperlinks

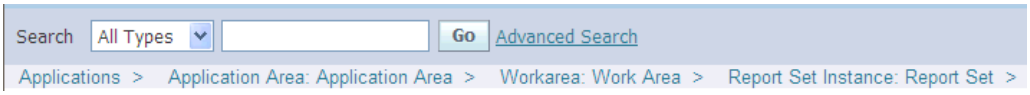
Click the name of any container or object to go to the Properties screen for that object.

Navigating Using Breadcrumbs

Breadcrumbs are a series of links to the screens you navigated through to arrive in your current location. They appear across the top of most Oracle Life Sciences Data Hub screens as shown in Figure 2-1.

Each of the screen names displayed in the breadcrumb path is a link. You can click it to return to that location.

Figure 2-1 Example of a Breadcrumb



> **Note:**
>
> Some breadcrumb links do not work; for example, after creating a new object. To recover from this situation, click the Applications tab at the top of the page and navigate as described in Navigating in the Applications Tab.

Using the Actions Drop-Down

The Actions drop-down list appears in the upper right corner of many screens in the Applications tab. The actions available in the list vary, depending on the context.

To use the Actions drop-down list, do the following:

- Click anywhere on the Actions field or arrow. The system displays the actions available in the current screen.
- Click the action you want to perform. The system displays the selected action in the field.
- Click **Go**. The system opens the appropriate screen.

Creating Domains, Application Areas, Work Areas, and Objects

Before creating Domains, Application Areas, and Work Areas, you should carefully design a set of these structures.

For details, see "Designing an Organizational Structure" in the *Oracle Life Sciences Data Hub Implementation Guide*. See also Keep Container and Object Names Short for Integrated Development Environments.

> **Note:**
>
> Oracle Life Sciences Data Hub can display a maximum of 200 rows at a time by default, so if you define more than 200 Domains within a Domain, or Application Areas within a Domain, or Work Areas within an Application Area, or objects within either an Application Area or Work Area, you get an error. Therefore, Oracle recommends that you design your organizational structure to avoid this problem. Alternatively, it is possible to reset the Oracle Applications profile FND: View Object Max Fetch Size to display more than 200 rows at a time; however this affects all your Oracle Applications.

- Creating Domains
  A Domain is the highest-level organizational structure in Oracle Life Sciences Data Hub. Domains are intended to contain logically related child Domains, Application Areas and/or a library of validated object definitions that are suitable for reuse.
- Creating Application Areas
  Application Areas contain Work Areas for the different life cycle stages of a single application. They also contain any object definitions that are created specifically for their application through one of their Work Areas.
- Creating Work Areas
  Application development work occurs in a Work Area.
- Creating or Adding Children, including Object Instances
  In Oracle Life Sciences Data Hub a "child" is an object that is contained in another object, which is called the "parent." In the main Applications screen you can use the Create Child icon to do the following:
- Creating Object Definitions
  Oracle recommends always creating objects in a Work Area, where you can install and test them.

Creating Domains

A Domain is the highest-level organizational structure in Oracle Life Sciences Data Hub. Domains are intended to contain logically related child Domains, Application Areas and/or a library of validated object definitions that are suitable for reuse.

See Object Definitions and Instances, and their Containers for further information.

To create a Domain, you must have the LSH Bootstrap Security application role. You must assign at least one user group to the Domain or no one will be able to use the Domain.

Nesting Domains

The number of levels of Domains contained within other Domains is determined by setting the Domain Nest Value profile for your Oracle Life Sciences Hub instance. If this value is set to 1, your top-level Domains cannot contain any child Domains. If this value is set to 2, your top-level Domains can contain child Domains. If the value is set to 3, the child Domains can contain child Domains, and so on.

See "Setting the Maximum Number of Nested Domains" in the *Oracle Life Sciences Data Hub System Administrator's Guide* for instructions.

- Creating a Top-Level Domain
  A top-level Domain has no parent in the user interface. You can create any number of Domains at the top hierarchical level.
- Creating a Child Domain
  A child Domain is a subdomain; a Domain contained in another Domain.

Creating a Top-Level Domain

A top-level Domain has no parent in the user interface. You can create any number of Domains at the top hierarchical level.

To create a top-level Domain:

1. Navigate to the Application Development home page.
2. Click **Maintain Domains**. The Maintain Domains screen appears.
3. Click **Create Domain**. The Create Domain screen appears.
4. Enter values in the following fields:
   a. **Name**. See Naming Objects.
   b. **Description**. See Creating and Using Object Descriptions.
   c. In the **Classification** section, select the following for the Domain:
5. In the **Classification** section, select the following for the Domain:
   a. **Subtype**. Select a subtype according to your company's policies.
   b. **Classification Values**. See Classifying Objects and Outputs for instructions.
6. Click **Apply**. The Domain screen appears.
7. **Forward Chain Enabled**:
   a. If checked, Programs and other executables in the Domain can be part of a forward chaining execution process if they are set up for it. This is the default value.
   b. If unchecked, objects in the Domain cannot be part of a forward chaining process even if they are set up for it.

   To change this setting, click **Update**, check or uncheck **ForwardChain Enabled**, and **Apply**.
8. From the **Actions** drop-down list, select **Apply Security** and click **Go**. The system opens the Manage Security screen.
9. Click **Assign Group**. The system opens the Search User Group screen.
10. If you know the name of the user group you want to assign to the Domain, enter it in the **Group Name** field. You can also enter part of a name and the system will return all groups that include the string you enter in their name. If you leave the field blank, the system returns all user groups.
11. Click **Go**. The system displays all user groups that satisfy your query.
12. Select any number of user groups: click their Select check box and click **Apply**.
13. Click **Return** to return to the Properties screen for the Domain.

Creating a Child Domain

A child Domain is a subdomain; a Domain contained in another Domain.

To create a Domain within a Domain, do the following:

1. On the main Applications screen, click the Create Child icon for the Domain in which you want to create a child Domain.
2. From the **Select Child** drop-down list, select Domain.
3. Click **OK**. The Create Domain screen opens.
4. Follow the same procedure as for Creating a Top-Level Domain, from Step 4 on.

Alternatively, do the following:

1. On the Applications Properties screen, click the name of the Domain in which you want to create a child Domain. The Properties screen for that Domain opens.
2. Click **Go**. The Create Domain screen opens.
3. Follow the same procedure as for Creating a Top-Level Domain, from Step 4 on.

Creating Application Areas

Application Areas contain Work Areas for the different life cycle stages of a single application. They also contain any object definitions that are created specifically for their application through one of their Work Areas.

To create an Application Area:

1. In a Domain, click **Create Application Area**. The Create Application Area screen appears.

   Alternatively, on the main Applications screen, click the Create Child icon next to the Domain name, then select Application Area and click **OK**.
2. Enter values in the following fields:
   a. **Name**. See Naming Objects.
   b. **Description**. See Creating and Using Object Descriptions
3. In the **Classification** section, select the following for the Application Area:
   a. **Subtype**. Select a subtype according to your company's policies.
   b. **Classification Values**. See Classifying Objects and Outputs for instructions.
4. Click **Apply**. The Application Area screen appears.
5. **Forward Chain Enabled**:
   a. If checked, Programs and other executables in the Application Area can be part of a forward chaining execution process if they are set up for it. This is the default value.
   b. If unchecked, objects in the Application Area cannot be part of a forward chaining process even if they are set up for it.

   To change this setting, click **Update**, check or uncheck **ForwardChain Enabled**, and **Apply**.

Creating Work Areas

Application development work occurs in a Work Area.

See also Using the Work Area Properties Screen.

To create a Work Area:

1. Do one of the following:
   a. In the Properties screen for Application Area, click **Create Work Area**.
   b. In the Application Development page, click the Create Child icon in the row of an Application Area.
1. The Create Work Area screen appears.
2. Enter values in the following fields:
   a. **Name**. See Naming Objects
   b. **Description**. See Creating and Using Object Descriptions.
3. Select the **Usage Intent** that represents the way you plan to use this Work Area. The options are: Development, Quality Control, and Production.
4. In the **Classification** section, select the following for the Work Area:
   a. **Subtype**. Select a subtype according to your company's policies.
   b. **Classification Values**. See Classifying Objects and Outputs for instructions.
5. Click **Apply**. The Work Area screen appears.
6. **Forward Chain Enabled**:
   a. If checked, Programs and other executables in the Application Area can be part of a forward chaining execution process if they are set up for it. This is the default value.
   b. If unchecked, objects in the Application Area cannot be part of a forward chaining process even if they are set up for it.

   To change this setting, click **Update**, check or uncheck **ForwardChain Enabled**, and **Apply**.

Creating or Adding Children, including Object Instances

In Oracle Life Sciences Data Hub a "child" is an object that is contained in another object, which is called the "parent." In the main Applications screen you can use the Create Child icon to do the following:

To add an object instance to a Work Area, do the following:

- Add a child Domain or Application Area to a Domain; see Creating a Child Domain and Creating Application Areas.
- Add a Work Area to an Application Area; Creating Work Areas.
- Add object instances to a Work Area (and simultaneously add a new definition to the parent Application Area, if you choose).

To add an object instance to a Work Area, do the following:

1. In the main Applications screen click the Create Child icon for the Workflow. The Create Child screen appears.

2. In the drop-down list, select the type of object you want to create.
3. Click **OK**. The system opens the relevant screen.

For information on creating each object type, see the following sections:

- **Business Area**. See Creating a Business Area.
- **Data Mart**. See Creating a Data Mart.
- **Load Set**. See Creating a Load Set.
- **Program**. See Creating a Program.
- **Report Set**. See Creating a Report Set.
- **Table**. See Creating a Table.
- **Work flow**. See Creating a Workflow.

Creating Object Definitions

Oracle recommends always creating objects in a Work Area, where you can install and test them.

If you create an object definition directly in a Domain or Application Area, you will not be able to test or use it until you create an instance of it in a Work Area and install it.

However, you can create an object definition directly in a Domain or Application Area library. See Creating Object Definitions.

Modifying Domains, Application Areas, and Work Areas

Domains, Application Areas, and Work Areas should be part of a carefully designed organizational structure; see "Designing an Organizational Structure" in the *Oracle Life Sciences Data Hub Implementation Guide*.

- **Name**. You can update the name at any time.
- **Description**. You can update the description at any time.
- **Usage Intent**. This property belongs to Work Areas only. See Changing a Work Area's Usage Intent.
- **Subtype**. Select a subtype according to your company's policies.
- **Classification Values**. You can change the classification values at any time. Use caution, however, because this may have the effect of changing the classifications of some or all objects contained in the Domain, Application Area, or Work Area, and all their children, grandchildren, and so on. See "Designing a Classification System" in the *Oracle Life Sciences Data Hub Implementation Guide.*

Usage Intent. Work Areas have one additional attribute that you can update.See Cloning Work Areas for Testing and Production for further information.

> **Note:**
>
> To change the usage intent, do not click **Update**. Instead, select **Upgrade Usage Intent** from the **Actions** drop-down list and click **Go**.

Duplicating, Removing, Moving and Promoting Domains

You must have special privileges to duplicate or remove a Domain.

- Duplicating or Copying a Domain
  To save time, you may want to set up one or more standard Domains with Application Areas, Work Areas, even object definitions and instances, and duplicate them.
- Removing a Domain
  Removing a Domain deletes the Domain and all the Application Areas, Work Areas, object definitions and object instances it contains.
- Moving a Domain
  If a Domain is nested in another Domain, you can move it to a different position in the same nest or to a different nest. The system checks that the Maximum Domain Nest Value will not be exceeded before performing the move operation. If the maximum value would be exceeded by the move, the system does not perform the move.
- Promoting a Domain
  If you have the necessary privileges you can promote a Domain; that is, move a Domain currently nested in another domain to the top level of the Applications hierarchy.

Duplicating or Copying a Domain

To save time, you may want to set up one or more standard Domains with Application Areas, Work Areas, even object definitions and instances, and duplicate them.

You can then modify the duplicate Domain; see Modifying Domains, Application Areas, and Work Areas.

The procedure is slightly different depending on where the Domain is. Top-level Domains are *duplicated*, and child Domains are *copied*. When you duplicate a top-level Domain, the system automatically creates the duplicate at the top level also.

When you copy a child Domain (or any other object) you can choose the target location for the copy. See Copying Objects for instructions.

To duplicate a top-level Domain, do the following:

- n the Applications Development screen, click **Maintain Domains**. The Maintain Domains screen opens, displaying all the Domains to which you have access.
- Select the Domain you want to duplicate by clicking the radio button next to it in the **Select** column.
  If you have access to many Domains you may need to click **Next 10** one or more times to find the Domain you want.
- Click **Duplicate**. The system creates a new Domain at the top level with the name `Copy of Original Domain Name`. You can find the new Domain listed alphabetically under C for Copy.

Removing a Domain

Removing a Domain deletes the Domain and all the Application Areas, Work Areas, object definitions and object instances it contains.

If you have the necessary security privileges, you can remove a Domain as you would any other object; see Removing Objects.

You can remove a Domain in either the Maintain Domains screen or in the main Applications screen.

Moving a Domain

If a Domain is nested in another Domain, you can move it to a different position in the same nest or to a different nest. The system checks that the Maximum Domain Nest Value will not be exceeded before performing the move operation. If the maximum value would be exceeded by the move, the system does not perform the move.

To move a Domain, do the following:

1. Go to the parent Domain of the Domain you want to move.
2. Select the Domain you want to move by clicking the **Select** check box.
3. Click the **Move** button. The Move Objects screen opens.
4. Navigate to the Domain into which you want to move the selected Domain. By default, the system displays the current Domain and all its children. You can select a different Domain using the Search field.
5. Click the **Select** radio button next to the Domain into which you want to paste the previously selected Domain.
6. Click **Apply**. The system removes the Domain from its previous location and puts it into the location you specified.

Promoting a Domain

If you have the necessary privileges you can promote a Domain; that is, move a Domain currently nested in another domain to the top level of the Applications hierarchy.

> **Note:**
>
> If the Domain currently inherits all its user group assignments from its parent Domain, you must make the following preparation before promoting it—explicitly assign those user groups to the domain that should have access to it in its new position at the top of the hierarchy. If you do not explicitly assign user groups before promoting it, only a user with the Bootstrap security administrator role will be able to see or work in the Domain after it is promoted.

To promote a Domain, do the following:

1. Go to the Properties screen of the Domain you want to promote.
2. From the **Actions** drop-down list, select **Promote Domain** and click **Go**.
3. The system promotes the Domain to the top level. You can no longer see it in the Applications Properties screen. Use the Search field to go to the Domain in its new location.

Managing Object Definitions

You can work on Oracle LSH object definitions within a Domain or Application Area from the Maintain Domain or Maintain Application Area screen.

For more information, see Object Definitions and Instances, and their Containers.

See also Removing Objects.

- Copying Object Definitions
  You can reuse Oracle LSH objects by copying their definitions from one container to another.
- Moving Object Definitions
  You can move Oracle LSH object definitions from one container to another.
- Creating Object Definitions
  You can create Oracle LSH object definitions inside a Domain or Application Area.

Copying Object Definitions

You can reuse Oracle LSH objects by copying their definitions from one container to another.

To copy one or more object definitions, do the following:

1. In the Application Development screen, click the icon in the **Manage Definitions** column for the Domain or Application Area from which you want to copy the object definition(s).

   You can also reach the Maintain Library screen by clicking the **Manage Definitions** button on the Domain or Application Area's Properties screen.
2. In the Maintain Library screen Oracle LSH objects are listed by object type. Click **Expand All** to see all the object definitions sorted by object type. To see object definitions of only one object type, click the **+** sign next to the object type.
3. Select the check box next to the object definition(s) you want to copy.
4. Click **Copy**. If the object definition(s) you selected to copy is/are currently checked out by you, the system shows a message requesting confirmation to check in the definition(s) before copying.

> **Note:**
>
> You cannot copy or move object definitions that are checked out by someone else.

5. Click **Yes**. The Copy Object(s) screen opens. Select the Domain or Application Area into which you want to paste the selected object definition(s). Follow instructions in Pasting Objects.

Moving Object Definitions

You can move Oracle LSH object definitions from one container to another.

To move one or more Oracle LSH object definitions, do the following:

1. In the Application Development screen, click the **Manage Definitions** icon next to the Domain or Application Area you want to copy the object definition(s) from.

   You can also reach the Maintain Library screen by clicking the **Manage Definitions** button on the Domain or Application Area's Properties screen.
2. In the Maintain Library screen Oracle LSH objects are listed by object type. Click **Expand All** to see all the object definitions sorted by object type. To see object definitions of only one object type, click the **+** sign next to the object type.
3. Select the check box next to the object definition(s) you want to move.
4. Click **Move**. If the object definition(s) you selected to move is/are currently checked out, the system shows a message requesting confirmation to check in the definition(s) before moving.

   > **Note:**
   >
   > You cannot copy or move object definitions that are checked out by someone else.

5. Click **Yes**. The Move Object(s) screen opens. Select the Domain or Application Area into which you want to paste the selected object definition(s). Follow instructions in Pasting Objects.

Creating Object Definitions

You can create Oracle LSH object definitions inside a Domain or Application Area.

> **Note:**
>
> In order to use or test changes to an object definition you must create and install an instance of it in a Work Area. See Creating or Adding Children, including Object Instances.

To create an Oracle LSH object definition, do the following:

1. In the Application Development screen, click the **Manage Definitions** icon next to the Domain or Application Area you want to create the object definition in.

   You can also reach the Maintain Library screen by clicking the **Manage Definitions** button on the Domain or Application Area's Properties screen.
2. Select the object type from the **Create** drop-down list. Click **Go**.

3. Enter values specific to the object type to create an object definition of that type.

3
Common Development Tasks

This section contains information on the following topics, which are tasks common to the definition of different types of Oracle Life Sciences Data Hub objects.

See also Getting Started.

- Creating and Reusing Objects
- Naming Objects
  Make an object name as descriptive as possible to help other users understand its purpose.
- Viewing Object Instances and Definitions
  The process of viewing and modifying object instances and definitions is similar for all object types.
- Understanding Object Versions and Checkin/Checkout
  This section includes the following topics.
- Upgrading Object Instances to a New Definition Version
  If an object instance references a definition that has been updated since the instance was created, you may want to update the instance so that it references the new version of the definition. You can upgrade object instances to a newer version of their source definition through the **Actions** drop-down list either from the definition or from the instance.
- Copying, Cloning, and Moving Objects
  This section contains the following topics.
- Removing Objects
  If you have the necessary privileges, you can remove an object instance from a Work Area or an object definition from an Application Area or Domain. After you remove an object, the system prepends its name with a tilde (~). You can then use the original name again within the same container.
- Classifying Objects and Outputs
  This section contains the following topics.
- Applying Security to Objects and Outputs
  In order to view or perform any other operation on an object, a user must belong to a user group assigned to the object. Objects automatically inherit the user group assignments of their immediate container.
- Validating Objects and Outputs
  This section includes the following topics.
- Reordering and Renumbering Objects
  You can use the Reorder function to change the order of some objects and the Renumber function to create sequential numbering for some objects.
- Defining and Mapping Table Descriptors
  This section contains the following topics.
- Creating, Modifying, and Submitting Execution Setups
  This section contains the following topics.

- Viewing Data
  This section contains the following topics.
- Viewing Jobs
  You can see a record of each submission of an executable object instance from the Jobs subtab on its Properties screen.
- Viewing All Outputs of a Program or Report Set
  You can view all existing outputs of executable objects several ways.
- Using the Actions Drop-Down List
  You can use the **Actions** drop-down list to modify an object's classifications, security user group assignments, validation status, and more.
- Defining Planned Outputs
  Different types of objects have different requirements for Planned Output definitions, which are placeholders for the actual outputs generated by the object during execution. See:

Creating and Reusing Objects

For information on creating new definitions, see the section for each object:

- Creating a New Table Definition and Instance; includes instructions for creating an Oracle LSH Table from a SAS data set
- Creating a New Program Definition and Instance
- Creating a New Load Set Definition and Instance
- Creating a New Report Set Definition and Instance
- Creating a New Data Mart Definition and Instance
- Creating a New Workflow Definition and Instance
- Creating a New Business Area Definition and Instance

This section contains the following topics. See also Naming Objects.

- Finding an Appropriate Definition
  You can look for an appropriate definition to use in several ways:
- Reusing Existing Definitions
  There are several ways you can reuse an existing definition.
- Creating an Instance of an Existing Definition
  Whenever possible, choose this option. It saves work by reducing the time required to define and validate objects and by promoting consistency across the application.
- Creating a New Definition and Instance
  If no object definition exists that suits your needs, or that you can modify to suit your needs, you must create a new one.
- Creating and Using Object Descriptions
  The description of an object definition should provide information to help Definers decide if they want to use the definition.

Finding an Appropriate Definition

You can look for an appropriate definition to use in several ways:

- If you know the location of the definition you need to use, click **Create an Instance of an Existing Definition** and use the Search utility there; see Using the Search and Select Window.
- To search with other criteria, including classifications and validation status, go to any screen with the Advanced Search utility, search, and make a note of the results. See "Advanced Search" in the *Oracle Life Sciences Data Hub User's Guide* for instructions.
- In the main Applications screen, navigate to a Domain or Application Area where you think you might find an appropriate definition, and click Maintain Definitions to see all the definitions in that Domain or Application Area library. See Navigating in the Applications Tab.

Reusing Existing Definitions

There are several ways you can reuse an existing definition.

- **Create an instance of an existing definition and use the definition as it is**. This is the easiest and fastest way to reuse a definition, requiring no revalidation of the definition. Use this method if the definition you need is validated and appropriate for your needs (see Creating an Instance of an Existing Definition).
- **Create an instance of an existing definition and then copy the definition to the current Application Area**. After you create an instance of an existing definition, check in the definition through the instance if it is not already checked in. Then check it out, selecting the option to create a copy of the definition in the current Application Area (see Checking Out an Object Definition through an Instance) and modify it as necessary.
  This method leaves the original definition unchanged and readily available for reuse elsewhere. You must validate the copied definition when you have finished modifying it.

  The system enforces unique object names within a container; see Naming Objects.
- **Create an instance of an existing definition and modify it as necessary**. If you have the necessary privileges, you can modify the definition through the new instance. The system creates a new version of the definition, which you must revalidate. You can label the new version to distinguish it from other versions; see Version Labels.

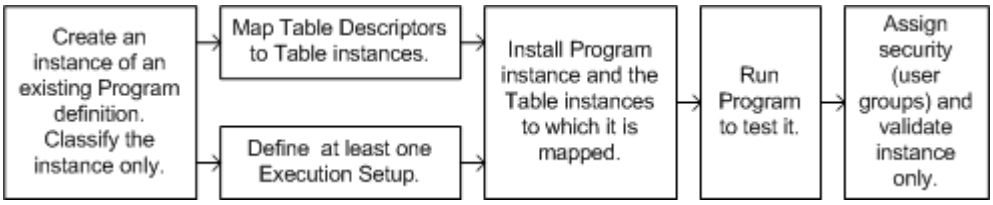
Creating an Instance of an Existing Definition

Whenever possible, choose this option. It saves work by reducing the time required to define and validate objects and by promoting consistency across the application.

See Finding an Appropriate Definition.

Figure 3-1 shows the process of creating an instance of an existing Program definition. The process is similar for all object types. When you create a new instance of an existing Program definition, you do not need to create Table Descriptors, Source Code, Parameters, or Planned Outputs because they are included in the Program definition. You do need to map the Table Descriptors to Table instances and define at least one Execution Setup. You then install the Program instance and the Table instances to which it is mapped, run and test the Program instance, validate it and assign user groups to it according to your company's policies.

Figure 3-1 Creating a New Program Instance from an Existing Program Definition



To create an Instance of an existing Definition:

1. Select **Create an Instance of an Existing Definition** and click **Apply**.

   The system displays the **Definition Source** field with a Search icon as shown here:

2. If you know the name of the object definition you want, enter it in the **Definition Source** field.

   If not, click the Search icon. The Search and Select pop-up window appears. See Using the Search and Select Window below.

3. Select a subtype and classifications for the new object instance, if necessary.
4. Click **Apply**. The system creates a new instance of the definition you specified in the current Work Area.
5. For all object types except Tables, you must map the Table Descriptors; see Mapping Table Descriptors to Table Instances for information.

   For all executable objects, you must create at least one Execution Setup; see Creating, Modifying, and Submitting Execution Setups.

- Using the Search and Select Window
  The Search and Select window is called from many screens in Oracle Life Sciences Data Hub. It receives its context from the calling screen. For example, if you click the Search icon in the Create Program screen, the Search and Select function finds only Program definitions.

Using the Search and Select Window

The Search and Select window is called from many screens in Oracle Life Sciences Data Hub. It receives its context from the calling screen. For example, if you click the Search icon in the Create Program screen, the Search and Select function finds only Program definitions.

Enter values in the following fields:

1. **Domain**. From the drop-down list, select the name of the Domain that contains the definition you need.

   If the Domain you specified contains one or more child Domains, you can see the child domain in the Domain drop-down list with the name of all parent domains preceding it and separated by a right angle bracket (>). For example:

Domain 1 > Child Domain 1 > Child Domain 2 > Child Domain 3

The system populates the **Application Area** drop-down list with the names of Application Areas contained in that Domain.

2. **Application Area**. If the definition is contained directly in a Domain library, do not select an Application Area.

   If the definition is contained in an Application Area, select the Application Area from the drop-down list.

3. **Search By**. If you know the name or version label of the object, select either **Name** or **Version Label** from the **Search By** drop-down list and enter either the name or version label in the field to the right. You can use special characters such as the % wildcard. For example, if you enter `Study_12345%` as a name, the system retrieves all objects whose name begins with Study_12345.

   > **Note:**
   >
   > For some object types this field is case-sensitive. If an object's name begins with STUDY_12345 and you enter `Study_12345%` the system does not retrieve the object because its name is not in uppercase.

4. **Display All Versions** If checked, the system returns all versions of the object with the name you specified in the **Search By** field. If unchecked, the system returns only the latest version. (If you specified Version Label in the Search By field, the system returns only the version with that label.)

5. **Display Not Null Version Labels** If checked, the system returns only object versions with a version label. This checkbox is relevant only if Name is selected in the Search By field above.

6. Click **Go**. The system displays the results of the search, including, for each definition retrieved, its name, version number, version label (if any), validation status, the username of the person who created the definition, and the date on which it was created. The system retrieves object definitions only if they are currently checked in.

   > **Note:**
   >
   > You can set your Preferences to make the system return only definitions with a validation status of Production.
   > You cannot create a new object instance based on a definition whose validation status is set to Retired.

7. Select the definition you want by clicking the icon in the Quick Select column, or by clicking its radio button and then clicking the **Select** button. The system returns you to the screen you came from, displaying the name of the object you selected.

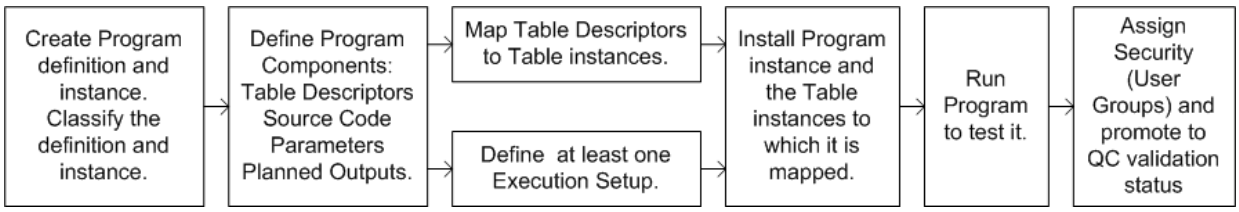
Creating a New Definition and Instance

If no object definition exists that suits your needs, or that you can modify to suit your needs, you must create a new one.

When you create a new definition and instance at the same time, the system simultaneously creates the definition in the current Application Area and an instance of it in the current Work Area. As you save your work, the system applies your changes to the appropriate object—definition or instance. The system applies the name and description you enter to both the definition and instance unless there is a conflict with either one; see Naming Objects.

Figure 3-2 shows the process of creating a new Program definition and instance. The process is similar for all object types. You first create the object as a whole, classify both the definition and the instance, then define component objects: in the case of Programs these include Table Descriptors, Source Code, Parameters, and Planned Outputs. All executable instances require mapping Table Descriptors to Table instances and defining at least one Execution Setup. You must **install** all object instances in their Work Area, assign user groups for object security and validate the definition and the instance according to your company's policies.

Figure 3-2 Creating a New Program Definition and Instance



Specific instructions for each object type are included in the section on each type:

- Creating a New Table Definition and Instance; includes instructions for creating an Oracle Life Sciences Data Hub Table from a SAS data set
- Creating a New Program Definition and Instance
- Creating a New Load Set Definition and Instance
- Creating a New Report Set Definition and Instance
- Creating a New Data Mart Definition and Instance
- Creating a New Workflow Definition and Instance
- Creating a New Business Area Definition and Instance

There are also instructions for creating the following component objects:

- Defining Parameters
- Defining and Mapping Table Descriptors
- Defining Source Code
- Defining Table Columns
- Defining Variables

Creating and Using Object Descriptions

The description of an object definition should provide information to help Definers decide if they want to use the definition.

You can search for words and phrases in the description using the Advanced Search feature.

Descriptions can contain up to 2000 characters.

Naming Objects

Make an object name as descriptive as possible to help other users understand its purpose.

Your company may develop naming conventions; see Customizable Naming Validation Package.

- Special Characters and Reserved Words
  To avoid problems, **do not use special characters** such as (& @ * $ | % ~ -) in object names, except for underscore (_). Also **do not use Oracle SQL or PL/SQL reserved words** in object names, especially the Oracle name.
- Duplicate Names: System Appends _1
  Oracle Life Sciences Data Hub enforces unique naming for each object type in the same container, whether the container is an organizational object—Domain, Application Area or Work Area—or a Report Set or Workflow. (Report Sets contain Programs. Workflows may contain Programs, Load Sets, Report Sets, and/or Data Marts.)
- Automatic Name Truncation
  Names can contain up to 200 characters. However, the system populates the values for Oracle Name and SAS Name with the value you enter for Name stored in uppercase, truncating the Oracle Name to 30 characters and the SAS Name to 32 characters.
- Keep Container and Object Names Short for Integrated Development Environments
  Although names can contain up to 200 characters, the maximum length of a file path is 256 characters in Windows.
- Naming Rules for Specific Object Types
  This section contains the following topics.
- Customizable Naming Validation Package
  Oracle Life Sciences Data Hub object creation and modification code includes a call to a predefined validation package from every object name field.

Special Characters and Reserved Words

To avoid problems, **do not use special characters** such as (& @ * $ | % ~ -) in object names, except for underscore (_). Also **do not use Oracle SQL or PL/SQL reserved words** in object names, especially the Oracle name.

For lists of reserved words, see:

Oracle® DatabaseSQL Language Reference at `https://docs.oracle.com/cd/E11882_01/server.112/e41084/toc.htm`

Oracle® Database PL/SQL Language Reference at `https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm`

For the latest information, you can generate a list of all keywords and reserved words with the V$RESERVED_WORDS view, described in the *Oracle® Database Reference* at https://docs.oracle.com/cd/E11882_01/server.112/e40402/toc.htm.

Duplicate Names: System Appends _1

Oracle Life Sciences Data Hub enforces unique naming for each object type in the same container, whether the container is an organizational object—Domain, Application Area or Work Area—or a Report Set or Workflow. (Report Sets contain Programs. Workflows may contain Programs, Load Sets, Report Sets, and/or Data Marts.)

The system enforces unique names only within the immediate container; for example, you cannot have two Program instances named Program_A in the same Workflow, but you can have a Program_A in a Workflow and another Program_A directly in the Application Area that contains the Workflow.

If you try to create a second object of the same type and name in the same container, the system creates the object but automatically appends an underscore and number 1 (_1) to the name. If you add a third object of the same type and name, the system increments the number (_2). For example, if you create a Program called Merge in Application Area 12345, and then create another Program called Merge in the same Application Area, the system names the new Program Merge_1. If you create a third Program called Merge in the same Application Area, the system names the new Program Merge_2.

Automatic Name Truncation

Names can contain up to 200 characters. However, the system populates the values for Oracle Name and SAS Name with the value you enter for Name stored in uppercase, truncating the Oracle Name to 30 characters and the SAS Name to 32 characters.

You can change the Oracle and SAS Names.

When you create a new object by uploading a table, view, data set, or column, the system truncates the Oracle Name and also replaces the last two characters with the number 01 or, if an object of the same type with the same name already exists in the same container, the next highest number (in this case, 02).

Keep Container and Object Names Short for Integrated Development Environments

Although names can contain up to 200 characters, the maximum length of a file path is 256 characters in Windows.

When you open an integrated development environment (IDE) such as SAS or run a SAS Program on your personal computer, Oracle Life Sciences Data Hub uses the actual full file path for Source Code definitions, Table instances, and the SAS runtime script. If the full file path exceeds this length you get an error and cannot open the IDE or run the Program. You can use a package to limit object name size or display an error when an object's file path is too long; see Customizable Naming Validation Package.

The full file path begins with the Oracle LSH username of the person who opens the IDE followed by the directory name `cdrwork`. It also includes all Subdomains in the path, if any. (The maximum number of Subdomains is 9. This number is configurable.)

- **Source Code definitions**. The path for Source Code definitions is: *username*>cdrwork>*Domain_name*>*Subdomain_name(s)* (0-9)>*Application_Area_name*>*Program_definition_name*>*Source_Code_definition_nameversion_number*>*fileref*>*source_code_filename*
- **Table instances**. The path for Table instances is: *username*>cdrwork>*Domain_name*>*Subdomain_name(s)* (0-9)>*Application_Area_name*>*Work_Area_name*>program instance name>program version>*Table_Descriptor_libname*> *Table_Descriptor_SAS_name*>
- **SAS runtime script**. The path for x is: *username*>cdrwork>*Domain_name*>*Subdomain_name(s)* (0-9)>*Application_Area_name*>*Work_Area_name*>*Program_instance_name*>*version_number*>setup

Naming Rules for Specific Object Types

This section contains the following topics.

- Naming Domains
  If you plan to export a Domain to another Oracle Life Sciences Data Hub instance, you may want to avoid using spaces in its name.
- Naming Source Code Objects
  There are important requirements for Source Code names:
- Naming Parameters
  Do not use spaces in the name of any Parameter you create for use in a Report Set.
- Naming OBIEE Business Areas
  Give a unique name to each Business Area that uses the same service location.

Naming Domains

If you plan to export a Domain to another Oracle Life Sciences Data Hub instance, you may want to avoid using spaces in its name.

Domain names with spaces must be entered with escape characters surrounding it in the Import Export Utility; for example: \" *domain name*\".

Naming Source Code Objects

There are important requirements for Source Code names:

- Source Code names must not include reserved words or special characters.
- Source Code names must include a file extension; for example, .sas for SAS, .rpt for Oracle Reports, or .sql for PL/SQL source code definitions. The system uses the name you enter to name the actual file that contains the source code.
- A Source Code's Oracle name must not be the same as the Oracle name of either a Table Descriptor or another Source Code in the same PL/SQL Program definition.

Naming Parameters

Do not use spaces in the name of any Parameter you create for use in a Report Set.

This will cause an error during Report Set postprocessing because the Parameter name becomes an HTML tag, and spaces are not allowed.

Naming OBIEE Business Areas

Give a unique name to each Business Area that uses the same service location.

Customizable Naming Validation Package

Oracle Life Sciences Data Hub object creation and modification code includes a call to a predefined validation package from every object name field.

By default, this package performs no validation and returns a value of TRUE, allowing users to enter any name in the field. However, you can customize the package to enforce your own naming conventions, full path length, or other object attribute standards. See "Customizing Object Validation Requirements" in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

Viewing Object Instances and Definitions

The process of viewing and modifying object instances and definitions is similar for all object types.

- Viewing Object Instances
  When you view an object instance (such as a Program or Table instance) in a Work Area, you see all the metadata details that apply to the object instance, including the properties that belong to the instance and the properties that belong to the definition.
- Viewing Object Definitions
  You can view object definitions two ways.

Viewing Object Instances

When you view an object instance (such as a Program or Table instance) in a Work Area, you see all the metadata details that apply to the object instance, including the properties that belong to the instance and the properties that belong to the definition.

The fields and settings in the upper portion of the screen (labeled "Instance Properties") belong to the object instance. The fields and settings in the lower portion of the screen (labeled "Definition Properties"), including the information under the subtabs in the middle of the screen, belong to the definition. The exception—for all objects that contain Table Descriptors—is Table Descriptor mappings, which belong to the instance but are displayed with the Table Descriptors, which belong to the definition.

Use the **Actions** drop-down list to see other characteristics of the instance, including its classifications, security user group assignments, validation supporting information, metadata reports, and Execution Setups.

Viewing Object Definitions

You can view object definitions two ways.

- Viewing Object Definitions in Domains and Application Areas
  To see an object definition where it is located, do the following:
- Viewing Object Definitions from Instances in a Work Area
  When you look at an object instance in a Work Area, you see many details of its source definition as well.

Viewing Object Definitions in Domains and Application Areas

To see an object definition where it is located, do the following:

1. In the main Applications screen, click the Domain or Application Area where the object you want to see is located.
2. Expand the relevant object type (such as Report Set or Program).
3. Click the link for the definition you want to see.

You can see old versions of object definitions in the Version History screen from the Actions drop-down list (see Version History).

You can also use Advanced Search to find old versions of objects or retired objects. See "Advanced Search" in the *Oracle Life Sciences Data Hub User's Guide*.

Viewing Object Definitions from Instances in a Work Area

When you look at an object instance in a Work Area, you see many details of its source definition as well.

The fields and settings in the lower portion of the screen (labeled "Definition Properties"), including the information under the subtabs in the middle of the screen, belong to the definition. The exception—for all objects that contain Table Descriptors—is Table Descriptor mappings, which belong to the instance but are displayed with the Table Descriptors, which belong to the definition.

In an object instance in a Work Area, you cannot see the source definition's classifications and user group assignments. To see these, you must go to the definition in its Domain or Application Area. You can do this directly from the Work Area as follows:

1. Click the Definition link in the Instance Properties section of the screen. The system opens the Properties screen of the definition in its Domain or Application Area.
2. Select the appropriate item—Apply Security or Assign Classification—from the **Actions** drop-down list and click **Go**. The system opens the appropriate screen.
3. To return to the object instance you came from, click the second last link in the breadcrumb at the top of the screen (see Navigating Using Breadcrumbs) or click **Return** to go back to the Definition page and click the last breadcrumb link from there.

Understanding Object Versions and Checkin/Checkout

This section includes the following topics.

- About Object Versions
  Oracle Life Sciences Data Hub keeps all object definitions and instances under version control, maintaining a record of the user who created each new version, the date and time the version was created, and, for definitions explicitly checked out, an optional comment entered by the person who checked it out.
- Checking Out Objects
  When you check out an object, either explicitly or implicitly, Oracle Life Sciences Data Hub creates a new version of it.
- Checking In Objects
  Oracle Life Sciences Data Hub finalizes an object version when it checks in the version, either explicitly or implicitly, as follows:
- Undoing Object Checkout
  After you explicitly check out an object definition, you can explicitly uncheck it. Oracle Life Sciences Data Hub discards the new version.
- Version Labels
  You can use version labels to identify important versions of objects.
- Version History
  You can see the version history of object definitions (not instances) by selecting **View Version History** from the **Actions** drop-down list.
- Versions of Component Objects
  Oracle Life Sciences Data Hub implicitly increments and tracks the version number of components of object definitions such as Source Code, Planned Outputs, Table Descriptors, and Columns that are not contained directly in a Domain or Application Area but only in other defined objects such as Programs or Tables.

About Object Versions

Oracle Life Sciences Data Hub keeps all object definitions and instances under version control, maintaining a record of the user who created each new version, the date and time the version was created, and, for definitions explicitly checked out, an optional comment entered by the person who checked it out.

The system creates a new version of an object when it is checked out, either explicitly or implicitly, and applies all changes to the new version until it is checked in. To make further changes you must check it out and modify the new version.

The system creates new object versions with the same user group assignments (security) and classifications as the previous version, but always gives new object versions a validation status of Development, no matter what validation status the previous version had.

When you create a new version of an object definition, instances of the previous version continue to point to the previous version. You can upgrade one or more instances to the new definition if you want to; see Upgrading One or More Instances from the Definition.

Report Sets' versioning is different from other objects'. Domains and Application Areas are not versioned; their version number is always 1.

Using Old Object Definition Versions: You can use old versions of object definitions as follows:

- Do not upgrade instances. Instances continue to point to the definition version selected when they were created until they are explicitly upgraded.
- When you create a new instance of an existing object definition, you can select any version of the definition after selecting "Display all Versions;" see Creating an Instance of an Existing Definition.
- You can copy an old version of an object definition at any time from its Version History screen; see Copying an Old Version of an Object Definition.
- If an object definition is not already checked out, you can check out an old version of it either from its Version History screen or through an instance that points to the old version.

> **Note:**
>
> If the definition is already checked out, you have the option to copy the version to the current Application Area.

> **Note:**
>
> When you check out an old version of an object, the system behaves the same way as when you check out the latest version. It creates a new version with the characteristics of the one you checked out and gives it a version number equal to the latest version number plus one.
> The system does not display the fact that the new version is based on an older version. If you want to make that clear, enter a checkout comment or create a version label with the information.

Checking Out Objects

When you check out an object, either explicitly or implicitly, Oracle Life Sciences Data Hub creates a new version of it.

- When you click **Update** to work on properties of an object definition or instance, Oracle Life Sciences Data Hub checks the object out.
- When you explicitly check out an object definition, Oracle Life Sciences Data Hub checks it out.
- When you explicitly check out an object definition through an instance of it, Oracle LSH implicitly checks out the instance and points it to the new version of the definition.
- When you create a new object definition or instance, Oracle Life Sciences Data Hub creates it as checked out by you, including when you choose to copy the definition and check it out (see Checking Out an Object Definition through an Instance.

While an object is checked out, only the person who initiated the checkout, either explicit or implicit, can modify the object , check it in, or uncheck it. That person can save changes multiple times. The exception is Report Sets, which allow multiple people to work on different sections at the same time.

> **Note:**
>
> If another user has already checked out an object definition, you cannot check out the object. However, you can copy the definition and modify the copy.

> **Note:**
>
> People with Checkin Administrator privileges can check in objects checked out by others.

- Checking Out an Object Definition through an Instance
  When you check out an object definition through an instance of it, you have the option to modify the definition in its current Domain or Application Area (if you have the required privileges) or to create a copy of the definition in the current Application Area, where you can modify it as necessary.
- Checking Out an Object Definition Directly
  To explicitly check out an object definition directly in its Domain or Application Area, do the following:
- Checking Out an Old Version of an Object Definition
  From the object definition's Version History page:

Checking Out an Object Definition through an Instance

When you check out an object definition through an instance of it, you have the option to modify the definition in its current Domain or Application Area (if you have the required privileges) or to create a copy of the definition in the current Application Area, where you can modify it as necessary.

The system points the instance to the new version of the definition whether you choose to modify it in its current location or copy the definition to the current Application Area.

To check out an object definition through an instance of it, do the following:

1. Navigate to the object instance.
2. Click **Check Out**. The Check Out screen appears.
3. Select one of the following:
   - **Check out existing definition**. This option allows you to modify the definition in its current location if you have the required privileges.
     If the definition is already checked out by another user, you get an error message when you select this option.
   - **Copy definition to the local Application Area and check out**. This option is useful if:

- you do not have the privileges required to modify the original definition in its location
- the current version of the definition should remain easily available for reuse
- the definition is checked out by another user

> **Note:**
>
> The system gives you this option even if the definition is already located in the current Application Area.

4. Click **Apply**.

Checking Out an Object Definition Directly

To explicitly check out an object definition directly in its Domain or Application Area, do the following:

1. Navigate to the object definition.
2. Click **Check Out**. The Check Out screen appears.
3. Type the reason for checking out the object in the **Comments** field.
4. Click **Apply**.

Checking Out an Old Version of an Object Definition

From the object definition's Version History page:

1. Navigate to the object definition in its Domain or Application Area, either in the Applications screen or by clicking the link to the definition in an instance of it.
2. From the **Actions** drop-down list, select **View Version History**. The Version History screen opens; see Version History.
3. Select the version you want to check out and click **Check Out**. The system creates a new version of the object based on the one you checked out, with a version number that is equal to the latest version's number plus one. All the changes you make apply to the new version.

Checking In Objects

Oracle Life Sciences Data Hub finalizes an object version when it checks in the version, either explicitly or implicitly, as follows:

- When you install or clone an object instance, Oracle Life Sciences Data Hub checks in the instance. It also checks in the underlying definition if it is not already checked in.
- After you explicitly check out an object definition, you can explicitly check it in.
- When you change the validation status of an object definition or instance, Oracle Life Sciences Data Hub checks it in if it is not already checked in.

To check in an object:

1. Navigate to the object's screen.
2. Click **Check In**. The Check In screen appears.
3. Type the reason for checking in the object in the **Comments** field.
4. Click **Apply**.

> **Note:**
>
> You can check in only those objects that you have checked out, unless you have the Checkin Administrator privileges.
> If someone else has checked out the definition and you do not have Checkin Administrator privileges, you cannot check it in. If you are working in an object instance, the username of the person who has checked out the object definition is displayed. If you are working directly on an object definition in an Application Area or Domain, you can find out who has checked out the object by running the All Instances report under Reports in the Actions drop-down list.

Undoing Object Checkout

After you explicitly check out an object definition, you can explicitly uncheck it. Oracle Life Sciences Data Hub discards the new version.

To undo a checkout for an object:

1. When you have checked out an object definition, the system displays a **Check In** button and an **Uncheck** button.
2. Click **Uncheck**. A confirmation for undoing the checkout appears.
3. Click **Yes**.

> **Note:**
>
> You can undo a check out for only those objects which you check in, even if you have the Checkin Administrator privileges.

Version Labels

You can use version labels to identify important versions of objects.

These labels are visible when you search for an object and when you upgrade object instances, and you can filter search results to return only object versions with labels.

To label an object version, follow these steps:

1. Navigate to the object version you want to label. To label an object definition, you must go to the definition in its Domain or Application Area.
2. From the **Actions** drop-down list, select **Version Label**. Click **Go**.
3. Do one of the following:

- Enter text for the label In the **Version Label** field.
- Click the Search icon. The system displays all labels that have been created for objects in the same location. Select one of them.

If you enter or select a label that has already been applied to a previous version of the same object, you receive a warning. If you apply the label to the current version, the system removes it from the previous version.

4. Click **Apply**. The system applies the label.

Version History

You can see the version history of object definitions (not instances) by selecting **View Version History** from the **Actions** drop-down list.

You can select any version of the object and click **Copy** or **Check Out** to copy it or check it out.

For each version you see the following information:

- **Name**. The object version name is also a hyperlink to the object version definition.
- **Description** of object version description.
- **Version** number.
- **Status**: Installable or Noninstallable.
- **Validation Status**: Development, Quality Control, Production or Retired.
- **Last Modified By**. Username of the person who created the version by modifying the object definition.
- **Last Modified**. The timestamp of the creation of the version.

If you click the + node you can see the following information:

- **Copied From**. If the object version was created by modifying the previous version of the object, that object version is displayed. If it was created by copying and pasting an object from another location, or by cloning a Work Area and all its objects, that information is displayed.
- **Version Label**. If a label was applied to the version, the system displays the label.
- **Check In Comments**. If the person who checked in the object version entered a comment, the system displays it here.
- **Check Out Comments**. If the person who created the object version by checking out the previous version entered a comment, the system displays it here.

Versions of Component Objects

Oracle Life Sciences Data Hub implicitly increments and tracks the version number of components of object definitions such as Source Code, Planned Outputs, Table Descriptors, and Columns that are not contained directly in a Domain or Application Area but only in other defined objects such as Programs or Tables.

To modify these objects you must check out their containing object. The system automatically increments their version number behind the scenes when you modify them within the containing object. When you check in the containing object, the system applies a new version number to the modified component object as well as the

containing object. If you uncheck the container, you effectively uncheck the component object as well and the system does not save the new version of either object.

The system increments the component's version number only if you modify it, so the version number of the component object may not match the version number of the containing object definition. Oracle Life Sciences Data Hub uses the information it stores about component versions to reconstruct past versions of the definition as a whole.

Component object definitions that are contained directly in a Domain or Application Area (Variables, Parameters, Parameter Sets, Notifications, and Overlay Templates) can have version labels (see Version Labels). They can also have classifications and user group assignments.

Upgrading Object Instances to a New Definition Version

If an object instance references a definition that has been updated since the instance was created, you may want to update the instance so that it references the new version of the definition. You can upgrade object instances to a newer version of their source definition through the **Actions** drop-down list either from the definition or from the instance.

For upgrading Source Code instances, see Upgrading Source Code And Undoing Source Code Upgrades.

- Upgrading One or More Instances from the Definition
- Upgrading to a Different Definition Version from an Instance
  To make an object instance point to a different version of its source definition, do the following:
- Upgrading to the Latest Version
  If you are working on an object instance that is not using the most current version of its source definition, the following buttons appear in the Definition Properties portion of the instance's screen:

Upgrading One or More Instances from the Definition

To upgrade one or more instances of a definition to a newer version of the definition, do the following:

1. Navigate to the definition in the Applications tab.
2. Check in the definition if it is not already checked in.
3. From the **Actions** drop-down list, select **Upgrade All Instances** and click **Go**. The system opens the Upgrade Instances screen with all instances of the definition displayed.

> **Note:**
>
> If you don't see the **Upgrade All Instances** option in the **Actions** drop-down list, then you are not in the latest version of the definition. You can upgrade instances only from the latest definition version.
>
> The exception to the above is if you are upgrading Source Code instances. See "Upgrading Multiple Source Code Instances".

For each instance, the system displays the following information:

- **Object Name**. The name of the object instance that points to any version of the definition.
- **Object Type**. The type of the object.
- **Object Version**. The version number of the instance.
- **Version Label**. The version label of the instance.
- **Installed Version**. The most recent version of the instance that was successfully installed.
- **Validation Status**. The instance's validation status.
- **Checked Out By**. The name of the person who has checked out the instance. If a person other than you has the instance checked out, then you cannot upgrade it: the check box next to it is grayed out.
- **Definition Version**. The version number of the definition to which the instance currently points.
- **Definition Validation Status**. The validation status of the definition.
- **Parent Name**. The name of the Parent object.
- **Parent Object Type**. The type of object that contains the instance.
- **Parent Validation Status**. The parent's validation status.
- **Container**. The container hierarchy for the instance.

4. Select one or more instances you want to upgrade. You can use the **Select All** and **Select None** functions and/or select or deselect instances individually by checking or unchecking their **Select** checkbox. Instances that already point to the current version of the definition cannot be selected.
5. Click **Upgrade**. The system changes the source definition of the selected instances to the new version of the definition.

> **Note:**
>
> If you find the **Upgrade** button not enabled, then the definition you want to upgrade to, is not checked in.

Upgrading to a Different Definition Version from an Instance

To make an object instance point to a different version of its source definition, do the following:

1. If the version of the definition to which you want to upgrade is not already checked in, navigate to it in its Domain or Application Area and check it in.
2. Navigate to the object instance in a Work Area.
3. From the **Actions** drop-down list, select **Upgrade Instance**.
4. Click **Go**. The system displays the available versions of the object definition in the lower portion of the screen.

   For each version of the definition, the system displays the following information:

   - **Version**. The version number of the definition version.
   - **Name**. The name of the definition version.
   - **Description**. The description entered for the definition version.
   - **Status**. The definition version's installation status.
   - **Validation Status**. The definition version's validation status.
   - **Version Label**. The label associated with the definition version, if any.
   - **Checked Out By**. If the definition version is checked out, the system displays the username of the person who checked it out. You cannot upgrade to a version that is checked out, and only the person who checked it out can check it in.
5. Click the icon in the **Upgrade** column for the version to which you want to point the instance. The icon of the version to which the instance is currently pointing is grayed out.

> **Note:**
>
> You can select an older version than the one to which the instance is currently pointing. This is useful if you want to undo an earlier upgrade.

Upgrading to the Latest Version

If you are working on an object instance that is not using the most current version of its source definition, the following buttons appear in the Definition Properties portion of the instance's screen:

- **View Latest**. Click to view the latest version of the definition.
- **Upgrade to Latest**. Click to upgrade to the latest version of the definition.

> **Note:**
>
> This option is available only if the definition is checked in, even if you are the person who has checked it out.

Copying, Cloning, and Moving Objects

This section contains the following topics.

- Copying Objects
  You can copy object definitions or instances from one location and paste them into another. You can copy a whole Work Area, Application Area, or Domain, with all its contents.
- Comparison of Copying and Cloning Individual Objects
  Cloning an object differs from copying an object as follows:
- Cloning Objects
  You can select one object instance to clone from either the Application Development screen or from the main Work Area screen. In the Work Area screen you can also select multiple objects and clone them at the same time to the same target Work Area.
- Moving Objects
  Moving an object removes it from its original location and puts it into a new location without breaking any references.
- Pasting Objects
  Pasting is the second part of a copy, move, or clone operation.

Copying Objects

You can copy object definitions or instances from one location and paste them into another. You can copy a whole Work Area, Application Area, or Domain, with all its contents.

When you copy a very large object—a Work Area, Application Area, Domain, Report Set, or Workflow—either by itself or as one of multiple selected objects, the system launches the copy operation as a batch job and displays a message with the Job ID.

When you copy an object instance or definition, the system copies only the current version (unless you explicitly select an older version in the View Version History screen; see Copying an Old Version of an Object Definition). When you paste the object into its new location, the system gives it a version number of one (1).

For additional information on the Copy operation, see Comparison of Copying and Cloning Individual Objects.

This section contains the following topics:

- Copying One Object Instance, Domain, Application Area, or Work Area
  From the main Applications screen you can copy a single object instance, Domain, Application Area, or Work Area.
- Copying One or More Objects at the Same Time
  You can copy multiple objects at once if they are all in the same location to begin with and you paste them all into the same location.
- Copying an Old Version of an Object Definition
  To copy an old version of an object definition, do the following:

Copying One Object Instance, Domain, Application Area, or Work Area

From the main Applications screen you can copy a single object instance, Domain, Application Area, or Work Area.

All links between objects that are included in the container object you copy are also copied. For example, if you copy a whole Application Area, and it includes an object instance in a Work Area that points to an object definition in the Application Area, the system copies the link, so that the copied instance in its new location points to the copied definition in its new location. If you copy a Work Area where Table instances are mapped to executable instances, the system copies the mappings so that the copied Table instances are mapped to the copied executable instances.

1. Click the Applications tab. The main Applications screen opens.
2. Select one object and click **Copy**. The system opens the Copy Objects screen and displays the objects you selected in the section labeled **Source—Source objects to be copied**.
3. Follow instructions for Pasting Objects.

Copying One or More Objects at the Same Time

You can copy multiple objects at once if they are all in the same location to begin with and you paste them all into the same location.

If you copy a Program instance and Table instance that are mapped to each other, the system automatically remaps them in the new location.

To copy and paste multiple object definitions or instances at the same time, do the following:

1. In the Applications tab, go to the Properties screen of the Domain, Application Area, or Work Area that contains the objects you want to copy or, for Domains and Application Areas, the Manage Definitions screen.
2. Select one or more objects and click **Copy**. The system opens the Copy Objects screen and displays the objects you selected in the section labeled **Source—Source objects to be copied**.
3. Follow instructions for Pasting Objects.

Copying an Old Version of an Object Definition

To copy an old version of an object definition, do the following:

1. Navigate to the object definition in its Domain or Application Area, either in the Applications screen or by clicking the link to the definition in an instance of it.
2. From the **Actions** drop-down list, select **View Version History**. The Version History screen opens; see Version History.
3. Select the version you want to check out and click **Copy**. The system opens the Copy Objects screen and displays the objects you selected in the section labeled **Source—Source objects to be copied**.
4. Follow instructions for Pasting Objects.

Comparison of Copying and Cloning Individual Objects

Cloning an object differs from copying an object as follows:

- **Instances and Definitions**. You can copy object definitions as well as object instances. You can clone only object instances.
- **Allowed Targets**. You can copy an object definition or instance into the same container—Domain, Application Area, or Work Area—that contains the source object. In that case, the Copy operation creates a copy of the object with the name Copy_of_*source_object_name*. You can clone an object instance only to a different Work Area; not the same Work Area that contains the source object instance.
- **Replacement.** If an object of the same type with the same name exists in the target Work Area, the Copy operation creates a new object in the target Work Area with a different name (_1 appended or final number incremented by 1). The Clone operation upgrades the target object with a duplicate of the source object (including the same name with nothing appended) unless the target is already identical to the source, in which case the Clone operation takes no action on the target object. Neither operation (Copy or Clone) compares the version number of the source and target objects (see **Version Number** below).
  If an object of the same type with the same name does **not** exist in the target Work Area, the Clone and Copy operations both create a duplicate of the source object, with the same name, in the target Work Area.
- **Implicit Cloning of Mappings and Table instances**. If you clone a Program, Load Set, Data Mart, Business Area, Report Set, or Workflow that is mapped to one or more Table instances in the same Work Area, the Clone operation automatically clones each Table instance and the mapping as well. If a source Table instance is in a different Work Area, the system copies the mapping to the same Table instance. The Copy operation does not copy any objects except those that are explicitly selected.
  If a Table instance with the same name already exists in the target Work Area of a Clone operation with implicit cloning of a Table instance, the system checks to see if the Table instance in the target Work Area was already cloned from the same source Table instance or not. If it was, the clone proceeds without warning and replaces the Table instance. If the target Table instance was originally defined in the target Work Area, or if it was cloned or copied from a different Table instance, you receive an error and cannot proceed with the cloning operation.

> **Note:**
>
> Cloning upgrades Table instances and all other instances; it does not delete data in Table instances unless the source Table instance has been modified in a destructive way, such as removing a Column. In that case, you get a warning when you make the destructive change that the change may prevent upgrading during the next installation. However, you do not receive a warning during the Clone operation.
> Changing a table's Blinding flag to Y from N is not considered a destructive change. Changing the Blinding flag from N to Y is allowed if there is no dummy data loaded.

- **Validation Status**. When you copy an object, the new object's validation status is always set to Development, no matter what the validation status of the source object was. When you clone an object, the new object has the same validation status as the source object.

  > **Note:**
  >
  > When you copy or clone an object, none of its validation supporting information links are maintained. They are maintained when you move an object.

- **Version Number**. When you copy an object, the new object's version number is always set to 1, no matter what the version number of the source object was. When you clone an object, the Clone operation increments the version number of the target object by 1.
- **Security**. The same privileges are required to copy and to clone objects: Read privileges for the object type in the source location and Create or Modify privileges for the object type in the target location.

For information about cloning an entire Work Area, see Cloning Work Areas for Testing and Production.

Cloning Objects

You can select one object instance to clone from either the Application Development screen or from the main Work Area screen. In the Work Area screen you can also select multiple objects and clone them at the same time to the same target Work Area.

See Comparison of Copying and Cloning Individual Objects for additional information about cloning object instances.

> **Note:**
>
> When you select a Program instance that is mapped to one or more Table instances in the same Work Area, Oracle Life Sciences Data Hub automatically clones the Table instance(s) and the mapping(s) as well as the Program instance.

This section contains the following topics:

- Comparison of Cloning Individual Objects and Whole Work Areas
  Cloning object instances is different from cloning a whole Work Area in the following ways:
- Cloning Objects from the Application Development Screen
  To clone an object from the Application Development screen, do the following:
- Cloning Objects from the Work Area Screen
  To clone one or more objects from the Work Area screen, do the following:

Comparison of Cloning Individual Objects and Whole Work Areas

Cloning object instances is different from cloning a whole Work Area in the following ways:

- **Implicit Cloning of Mapped Table Instances**. If you clone an individual executable object—Program, Load Set, Data Mart, Business Area, Report Set, or Workflow—that is mapped to one or more Table instances in the same Work Area, the Clone operation automatically clones the Table instance and the mapping as well.
  If you clone a whole Work Area, the system replaces any Table instances in the target Work Area that have been modified in the source Work Area since the last clone.
- **Checks**. When you clone a whole Work Area, the system duplicates the source Work Area, resulting in an identical target Work Area. Any object instances present in the target Work Area before the clone are either dropped or replaced, and objects that were not previously present are created.

  When you clone an object instance, the system does the following before performing the clone:

  – **Checks Table instance origin during implicit clone**. When you clone an individual Table instance implicitly by cloning an executable to which it is mapped, and a Table instance with the same name already exists in the target Work Area, the system checks to see if the Table instance in the target Work Area was already cloned from the same source Table instance or not. If it was, the clone proceeds without warning and replaces the Table instance. If the target Table instance was originally defined in the target Work Area, or if it was cloned or copied from a different Table instance, you receive a warning and cannot proceed with the Clone operation.

  > **Note:**
  >
  > The system performs this check only when you clone a Table instance implicitly. If you explicitly select a Table instance for cloning, the system does not make this check: if a Table instance already exists in the target Work Area with the same name, the new Table instance replaces it, regardless of the target Table instance's origin. After you have cloned a Table instance from two different source Table instances, the system allows you to clone the Table instance implicitly as well as explicitly from either source.

  – **Prevents multiple executables from writing to the same Table instance**. When you clone an individual Table instance, either explicitly or implicitly when you clone a Program or Load Set instance to which it is mapped, the system does not allow the Clone operation to succeed if the result would be that two executables (Programs or Load Sets) wrote to the same target Table instance. When you clone a whole Work Area, this is not an issue because the system has already prevented multiple executables from writing to the same Table instance in the source Work Area.

- When you clone a whole Work Area, you must apply a label to the source and target Work Areas. However, when you clone an object, you cannot apply a clone label to the source and target objects.
- When you clone a whole Work Area you can update the Usage Intent of the target Work Area as part of the Clone operation.
- When you clone an individual object instance the source Work Area is not checked in as part of the cloning operation as it is when you clone a whole Work Area.
- When you clone an individual object instance you cannot create a new target Work Area as part of the cloning operation, as you can when you clone a whole Work Area and specify an Application Area as the target.

Cloning Objects from the Application Development Screen

To clone an object from the Application Development screen, do the following:

1. Click the Applications tab to open the Application Development screen.
2. Expand the Application Area and Work Area that contain the object you want to clone.
3. Click the icon in the Clone column on the same row as the object you want to clone. The Clone Instances screen opens with the selected object displayed in the **Instance Objects to Be Cloned** section.
4. In the **Clone Destination** section, select the Work Area in which you want to create a clone of the object.
    a. If you want to clone the object into a Work Area in a different Domain, click the Search icon and follow instructions at Select a Domain on the Applications Screen. The system displays the Application Areas contained in the Domain.
    b. Expand the node (+) of the Application Area that contains the target Work Area.
    c. Select the target Work Area.
5. Click **Apply**.

Cloning Objects from the Work Area Screen

To clone one or more objects from the Work Area screen, do the following:

1. In the Application Development screen, click the hyperlink in the Name column of the Work Area that contains the object or objects you want to clone. The Work Area Properties screen opens.
2. Select each object you want to copy and click **Clone**. The Clone Instances screen opens with the selected object(s) displayed in the **Instance Objects to Be Cloned** section.
3. In the **Clone Destination** section, select the Work Area in which you want to create a clone of the object.
    a. If you want to clone the object into a Work Area in a different Domain, click the Search icon and follow instructions at Select a Domain on the Applications Screen. The system displays the Application Areas contained in the Domain.
    b. Expand the node (+) of the Application Area that contains the target Work Area.

c. Select the target Work Area.

4. Click **Apply**.

Moving Objects

Moving an object removes it from its original location and puts it into a new location without breaking any references.

That is, instances of a definition being moved continue to point to the definition in its new location. The system moves all versions of the object together. The Move operation is available inside Domains, Application Areas, and Report Sets for object definitions and Report Set Entries, respectively. It is not available in Work Areas for object instances.

You can move multiple objects at once if they are all in the same location to begin with and you paste them all into the same new location.

You can move an object to a different location if you have Create privileges for that object type in that location and Modify privileges on its parent object (Domain, Application Area, or Report Set definition).

Use the Move operation to do the following:

- Move object definitions from Application Areas to Domains when they have been thoroughly tested and approved for reuse.
- Move an Application Area or child Domain, with all its contents, to a different Domain.
- Move object definitions from one Application Area or Domain to another.

To move one or more objects, do the following:

1. Navigate to one of the following locations:
    - To move one or more object definitions: In the Applications tab, click **Manage Definitions** for the Domain or Application Area from which you want to move objects. In the Maintain Library screen that opens, select the object you want to move and click **Move**.
    - To move one ore more Application Areas and/or child Domains: In the Applications tab, click the Domain's hyperlink in the **Name** column. In the Domain properties screen that opens, select one or more Application Areas and child Domains and click **Move**.
    - To move a child Domain or Application Area, one at a time: In the Applications tab, Application Development screen, click the icon in the object's **Move** column.
2. If you are moving objects (not Application Areas or Domains) a confirmation message appears, listing the objects you have selected to move that are currently checked out, if any, and telling you that they will be checked in as part of the Move operation.

    Click **Yes** to continue with the operation. The Move Objects screen opens.
3. Select the new location: follow instructions for Pasting Objects.

Pasting Objects

Pasting is the second part of a copy, move, or clone operation.

The objects you have selected appear in the upper portion of the screen.

1. In the section labeled **Destination—Target for objects when paste applied**, select the Domain into which you want to paste the object(s).

   The current Domain is displayed by default. You can enter the name of another Domain or click the Search icon and select another Domain; see Using the Search and Select Window.
   The system displays the Domain you selected and any child Domains and Application Areas it contains. You can click the + nodes to see Work Areas and objects within the Work Areas.

2. Click the Select radio button for the Domain, Application Area, Work Area, Report Set or Workflow into which you want to paste the object or objects.

   The system activates the radio button only for valid locations for the object type(s) you have selected.

3. Click **Apply**. The system creates version one (1) of each object in the new location.

> **Note:**
>
> If the target location already contains an object of the same type and with the same name, the system appends _1 to the name of the object you copy into the target location. If the object with the same name already ends in a number, the system renames the copied object with the number incremented by one (1).

Removing Objects

If you have the necessary privileges, you can remove an object instance from a Work Area or an object definition from an Application Area or Domain. After you remove an object, the system prepends its name with a tilde (~). You can then use the original name again within the same container.

The following rules apply:

- Object metadata is never completely removed. The system gives the definition or instance an end date but stores a record of it as it existed during the time it existed. However, when a Table instance is removed, all its data is deleted.
- You cannot remove an object instance that has been installed and executed.
- You cannot remove an object definition that is used as a source definition by one or more instances. You must remove the instance(s) first. If you try to remove an object definition with an instance, you receive an error message with the name of the instance object. To proactively check if a definition has instances, select **Upgrade Instances** from the **Actions** drop-down. The system displays all instances with their locations, validation statuses, and other information.

- You cannot remove an object that contains objects that have not been deleted. This applies to Domains, Application Areas, Work Areas, Report Sets, and Workflows.

To remove objects:

1. In the Applications tab, go to the location of the object you need to remove.

> **Note:**
>
> In the main Applications screen you can remove only one object at a time. However, if you go to the Properties screen for a Domain, Application Area, or Work Area, or the Manage Definitions screen of a Domain or Application Area, you can remove multiple objects at a time.

2. Select one or more objects to remove.
3. Click **Remove**. The system gives you a message asking you to confirm that you want to remove the selected object(s).
4. Click **Yes**. The system removes the object or objects and returns you to the Application Development screen.

- Rules for Removing Objects
  Oracle Life Sciences Data Hub enforces the following rules for object removal.

Rules for Removing Objects

Oracle Life Sciences Data Hub enforces the following rules for object removal.

- You cannot remove an object if other objects are dependent on it. For example, you cannot remove an object definition if there are one or more instances of it. (You can set the object definition's status to Retired, however, and then no additional instances of it can be created.)
- You cannot remove Table instances with a validation status of Production. This is to protect production data.
- In some cases the removed object is still displayed in the user interface. The system then renames the object by prepending a tilde (~) to its name.
  Using a tilde to rename the object signifies that the object has been removed and allows you to create a new object of the same type in the same container with the same name.

  Examples:

  – If an executable object has been executed and then removed, it appears in the Job Details page for the job preceded by a tilde (~).

  – You can remove a component object (such as a Parameter) from the object that contains it (such as a Program) or an executable object (such as a Program) from either a Report Set or a Workflow. In these cases Oracle LSH continues to display the removed object in the Version History screen for its container but renames the removed object by prepending one or more tildes (~) to its name in previous versions of the container object. You can then create a new object of the same type with the same name in the latest version of the same container. The old versions of the container continue to function as before.

Classifying Objects and Outputs

This section contains the following topics.

- About Classification
  Classifications are logically related labels defined by your company that help people find the objects you define and outputs they generate:
- Classifying Objects
  To classify an output, follow these instructions.
- Classifying Outputs
  Classifying outputs is different from classifying other objects because you must classify them before they are created. You can reclassify them after they are created if you have the necessary privileges.

About Classification

Classifications are logically related labels defined by your company that help people find the objects you define and outputs they generate:

- Oracle Life Sciences Data Hub displays outputs and submission forms (Execution Setups) by their classifications in the Reports tab.
- The Advanced Search feature allows people to search for object definitions, instances, and outputs by their classifications.

When you create a new object, you must select a subtype for it according to your company's policies. The subtype determines which classification hierarchy levels you can or must use to classify your object.

There may be default or inherited classification values assigned to your object. You can accept these or override them with values you select, according to your company's policies.

Example: Your company has set up a classification hierarchy with three levels, Project, Study, and Site. The Project level values include all the therapeutic areas with current trials. The Study level values include all the current trials, each linked to its therapeutic area. The Site level values include all the sites participating in each trial, each linked to its trial or trials. A given subtype may require classification only at the Study level.

When you classify a Program instance, for example, that will be used in all sites for Trial X, you classify it at the Study level to Trial X. If the whole Application Area where the Program is located is devoted to Trial X, your company may have set up its classification system so that all objects within the Application Area automatically inherit the classification value Trial X from the Application Area. This may be true for Work Areas within the Application Area and the object instances in the Work Areas as well.

Reclassification: You can reclassify an object definition or instance at any time.

To view or modify the classification of the definition, you must view it in its location—Application Area or Domain. You cannot see the definition's classifications from an instance of it in a Work Area.

Object Versions and Classifications: Classifications apply to all versions of an object. When you reclassify an object, either explicitly or through inheritance, the new classifications apply to all versions of the object.

Classifying Objects

To classify an output, follow these instructions.

For further information about classifying outputs, see Classifying Outputs.

To classify an object instance or definition:

1. When you first create an object, a classification interface appears automatically. If you are creating a new object definition and instance at the same time, a classification interface appears for both the definition and the instance.

   If you want to change or view the classifications afterward, go to the object and select **Assign Classification** from the **Actions** drop-down list. To change the classifications, click **Update**.

2. Select a **Subtype** from the drop-down list, according to your company's policy, and click **Go**.

   The system displays the classification hierarchies and levels defined for that subtype. Under each hierarchy name, the system displays the levels of that hierarchy for which objects of the subtype you specified must have a value. For example, in the Project/Study/Site hierarchy, an object subtype may require classification at the Study level, while classification at the Site level is optional. If classification at the Project level is not predefined for the object subtype, the Project level is not displayed.
   For each level you see the following information:

   - **Classification**. The system displays the name of the level inside its hierarchy.
   - **Type**. The system displays whether the row shows a Subtype, a Hierarchy, a Level, or a Term. For terms, this field does not say "Term" but displays Inherited or Explicit, as the case may be.
   - **Assignment Type**. If the classification type is **Inherited**, the object is automatically classified to the same value for this level as its container object. For example, an object definition inherits the value from its Application Area or Domain, and an object instance inherits the value from its Work Area.
     If the classification type is **Explicit**, you can enter one or more values. There may be a default value; if so, you can override it.

     You can change the type. Normally it is best to leave inherited classifications alone, but there may be times when a particular object should not inherit the classification value of its parent.
   - **Search and Add Value**. If the assignment type is Explicit, you can search for values and assign them to the object.
   - **Mandatory**. If **Yes**, objects of this subtype must have a value assigned for this level, either explicitly or inherited. If **No**, you can assign a value from this level to the object, but it is not required.

3. For Explicit-type assignments, you can search and add values, as follows. If classification to the level is mandatory, you must do so.

- Click the + icon in the **Search and Add Value** column. The system opens the Select Classification Hierarchy Terms/Values screen with a field displayed for each level in the hierarchy.
- You can click **Go** to see all the possible classifications at once, or you can narrow the search by specifying a value in one or more of the fields. For example, in the Project/Study/Site hierarchy, if you enter Project A in the Project field and then click **Go**, the system returns only studies and sites related to Project A.
- Select one or more values in the Results section and click **Select and Continue**. The system returns to the Assign Classification screen with the value or values you select displayed under their classification level.

4. Repeat for each level for which you want to assign classification values.
5. Click **Apply**. The system saves all the classification assignments you have made for the object.

> **Note:**
>
> The system applies classifications to objects by running the Oracle LSH Context Index Refresh program every two minutes. Therefore you may have to wait up to two minutes for your classifications to take effect.

Classifying Outputs

Classifying outputs is different from classifying other objects because you must classify them before they are created. You can reclassify them after they are created if you have the necessary privileges.

Report Sets have a single set of classifications for the Report Set as a whole. If Programs contained in a Report Set have classifications, they have no effect.

Workflows do not have a Planned Output for the Workflow as a whole. Outputs generated by Programs and other executables in a Workflow are classified according to the Planned Output definition in the Program or other executable.

Planned Outputs: An output is generated by running an executable object instance — Program, Report Set, Load Set, or Data Mart—that has a Planned Output defined as as a placeholder for the actual output. You classify the output by classifying the Planned Output. The process for classifying a Planned Output is the same as classifying any other object, except that you have an additional option: the Program itself can create the classification at runtime (Classification by Parameter Value).

If you assign more than one classification value to a particular Planned Output, the actual output appears in multiple places in the Reports screen.

Execution Setups: If you set the Planned Outputs's assignment type for a particular classification level to Inherited, the actual output inherits the value for that level from the Execution Setup.

Since the Execution Setup belongs to the Program, Load Set, Data Mart, or Report Set instance (unlike a Planned Output, which belongs to the definition), this approach allows you to classify the outputs of different instances of the same definition in different ways.

Classification by Parameter Value: You can arrange for an output to be classified at runtime by specifying a classification type of Parameter and specifying the Parameter whose value is used as the classification assignment.

The Parameter can be an Input/Output Parameter visible, settable, and required in the Execution Setup, so that the person submitting the Program or Report Set must enter a value.

Alternatively, in the case of a Program, it can be an Output type Parameter and the Program logic must populate its value with a valid classification term.

You can define a list of values for Parameters based on classification levels; see Defining Allowed Values for further information.

- Classifying Outputs Before They Are Generated
  The process is the same as for classifying object definitions and instances, except that an additional option is available for outputs.
- Classifying Outputs After They Are Generated
  After you have run a Program, Report Set, Load Set, or Data Mart and created an output, you can change its classifications if you have the necessary privileges.

Classifying Outputs Before They Are Generated

The process is the same as for classifying object definitions and instances, except that an additional option is available for outputs.

1. Go to the object—Program, Data Mart, Load Set, or Report Set—that will generate the output you want to classify.
2. Click the **Planned Outputs** tab.
3. Click the link of the Planned Output you want to classify. The Planned Output Properties screen opens.
4. Select **Assign Classification** from the **Actions** drop-down list and click **Go**. The Classification screen opens.
5. Classify the Planned Output. Follow instructions at Classifying Objects and Outputs. However, in Programs and Report Sets you have one additional option: an assignment type of Parameter (Classification by Parameter Value).

   If you select **Parameter** as the **Assignment Type**, the system adds another column called **Parameter instance** and populates a drop-down list in that column for the row with the names of all the Parameter instances in the Program or Report Set instance. Select a Parameter that will return a valid value for that classification level.

Classifying Outputs After They Are Generated

After you have run a Program, Report Set, Load Set, or Data Mart and created an output, you can change its classifications if you have the necessary privileges.

Navigate to the output in Oracle Life Sciences Data Hub. There are several ways to do this:

- **My Home**. In the **Job Execution** section at the bottom of the My Home page, click the **Job ID** of the job that produced the output. The Job Execution Details screen opens. Click the hyperlink that is the output's name.

You can see outputs through the My Home page only if you submitted them yourself.

- **Reports**. In the Reports tab, navigate to the output using its current classifications, then click on the icon in the **Action** column.
- **Administration**. In the Job Execution subtab of the Administration tab, query for the job that produced the output. See "Querying for Jobs" in the *Oracle Life Sciences Data Hub System Administrator's Guide*. The system returns the search results. Click the **Job ID** of the job that produced the output, then click the hyperlink that is the output's name.
  If you have access to the Administration tab, you can see all users' jobs.

> **Note:**
>
> You can use the Advanced Search feature from most screens to find outputs.

To reclassify the output, do the following:

1. In the Output Properties screen, select **Assign Classification** from the **Actions** drop-down list. The Assign Classification screen opens.
2. Change the classifications as necessary; see Classifying Objects.

Applying Security to Objects and Outputs

In order to view or perform any other operation on an object, a user must belong to a user group assigned to the object. Objects automatically inherit the user group assignments of their immediate container.

Therefore, by default, any object you create has the same user group assignments as its immediate container (Work Area, Application Area, Domain, Report Set or Workflow).

If you have the necessary privileges you can accept these default assignments or explicitly de-assign any of the user groups assigned to your object. You can also add other user group assignments. The changes you make will be inherited by objects contained by your object, if any.

- Viewing User Group Assignments
  To view user group assignments for an object:
- Assigning User Groups to an Object
  To assign a new user group:
- Removing User Group Assignments
  To remove a user group assignment:
- Reassigning a User Group as Inherited
  To re-assign a user group whose inherited assignment was revoked, do the following.

Viewing User Group Assignments

To view user group assignments for an object:

1. Go to the Properties screen for the object.
2. From the **Actions** drop-down, select **Apply Security**. The Manage Security screen opens.

   Each user group currently assigned to the object is displayed with its assignment status, which reflects the way it was assigned to the object:

   - **Assigned**. The user group was explicitly assigned to the object.
   - **Inherited**. The object inherited the user group assignment from its containing object.
3. Click **Return** to return to the object's Properties screen.

Assigning User Groups to an Object

To assign a new user group:

1. Go to the Properties screen for the object.
2. From the **Actions** drop-down, select **Apply Security**. The Manage Security screen opens.
3. Click **Assign Group**. The Search User Group screen appears.
4. In the **Group Name** field, enter the name of the user group you want to add, if you know it. If you do not know it, you can click **Search** to retrieve all user groups, or enter part of the name plus % and click **Search**.

   The system displays the search results.
5. Select one or more groups by checking the **Select** checkbox and clicking the **Apply** button.

   The system returns you to the Manage Security screen with the newly assigned user group(s) displayed.

Removing User Group Assignments

To remove a user group assignment:

1. Go to the Properties screen for the object.
2. From the **Actions** drop-down, select **Apply Security**. The Manage Security screen opens.
   - Click the icon in the **Un-Assign** column if the group's status is Assigned.
   - Click the icon in the **Revoke** column if the group's status is Inherited.
3. Click **Yes** when asked to confirm your action of unassigning or revoking the selected user group from the selected object.
4. Click **Return** to return to the object's Properties screen.

Reassigning a User Group as Inherited

To re-assign a user group whose inherited assignment was revoked, do the following.

This operation restores the assignment as inherited; if the user group is revoked from the parent object, it is automatically revoked from the current object as well.

1. Go to the Properties screen for the object.
2. From the **Actions** drop-down, select **Apply Security**. The Manage Security screen opens.
3. Click the icon in the **Un-Revoke** column.

> **Note:**
>
> It is possible to explicitly assign a user group that is already assigned by inheritance. To unassign such a group, you must undo both of its assignments: first Un-Assign and then Revoke.

Validating Objects and Outputs

This section includes the following topics.

- About Object Validation
  Your organization must develop standards for testing Oracle Life Sciences Data Hub defined objects to demonstrate that they are valid; that is, that they handle data as they are designed to do, without introducing any corruption.
- About Output Validation
  By default, outputs inherit the validation status of their Execution Setup. The Execution Setup inherits its validation status from its executable instance. An instance cannot have a higher validation status than its source definition.
- Validation Statuses
  There are four validation statuses in Oracle Life Sciences Data Hub: Development, Quality Control, Production, and Retired. Your organization must develop a policy on the use of validation statuses.
- Changing Objects' Validation Status
  Object definitions and instances both have a validation status.
- Changing Outputs' Validation Status
  To change an output's validation status, do the following:
- Adding Supporting Information
  Before you promote an object to the next validation status, your organization may require that you provide documentation about the object: either a link to an output generated by the object, such as a report or log file, or a document such as a functional requirements document for the object.
- Validation Rules
  The system enforces the following rules based on objects' validation status.

About Object Validation

Your organization must develop standards for testing Oracle Life Sciences Data Hub defined objects to demonstrate that they are valid; that is, that they handle data as they are designed to do, without introducing any corruption.

As you create object definitions and instances you must follow your organization's policy for validating them in compliance with industry regulations.

All Oracle Life Sciences Data Hub objects that are contained directly in a Domain, Application Area, or Work Area—including definitions and instances of Tables, Programs, Load Sets, Data Marts, Report Sets, Workflows, and Business Areas, as well as Variables, Parameters, and Notifications—have an attribute called Validation Status with possible values Development, Quality Control, Production, and Retired.

To support object validation, Oracle Life Sciences Data Hub provides the following tools:

- **Validation Status**. When an object definition or instance meets the standards set by your organization for a higher validation status, you can change its status. See Validation Statuses for further information.
- **Validation Supporting Information**. You can link an object with outputs and documents. See Adding Supporting Information for further information.
- **Report Coversheets**. You can generate a coversheet for every report output that displays all the defined objects that interacted with the data in the report from when it was loaded into Oracle Life Sciences Data Hub to when it was reported out of Oracle Life Sciences Data Hub, including the validation status of each object. See "Generating a Coversheet with Validation and Data Currency Information" in the *Oracle Life Sciences Data Hub User's Guide* for further information.
- **Work Area Usage Intent** and **Cloning**. Work Areas have an attribute called Usage Intent with the same list of values (except Retired) as the Validation Status attribute: Development, Quality Control, Production. Oracle LSH enforces Validation Rules based on the interaction of Work Area usage intent and object validation statuses. Work Areas also have a special operation called cloning that enables you to create clean, distinct environments for testing objects and for production; see Cloning Work Areas for Testing and Production.

Special rules apply to the validation of Report Set Entries, which you can validate independently from other Report Set Entries and the Report Set as a whole. See Validating Report Set Definitions and Outputs for further information.

About Output Validation

By default, outputs inherit the validation status of their Execution Setup. The Execution Setup inherits its validation status from its executable instance. An instance cannot have a higher validation status than its source definition.

Therefore, by default an output cannot have a validation status of Production unless the Program (or other) definition used to produce it has a validation status of Production.

You can manually change the validation status of an output after it is generated; see Changing Objects' Validation Status.

For Programs only, you can set a flag in the definition to force the system to set the validation status of an output to Development regardless of the validation status of the Program definition, instance, or Execution Setup. You can then upgrade the output's validation status manually according to your company's policies. For information about using this approach in Report Sets see Validating Report Set Definitions and Outputs.

Validation Statuses

There are four validation statuses in Oracle Life Sciences Data Hub: Development, Quality Control, Production, and Retired. Your organization must develop a policy on the use of validation statuses.

Oracle Life Sciences Data Hub enforces some behavior based on validation statuses; see Validation Rules.

Development: The Development status is intended for objects that are being developed and unit tested. All new objects have a validation status of Development when they are created.

Quality Control: The Quality Control status is intended for objects that are ready for or undergoing formal testing.

Production: The Production status is intended of objects that are ready for or are being used in a production environment.

Retired: The Retired status is for objects you no longer wish to use.

Changing Objects' Validation Status

Object definitions and instances both have a validation status.

You must work directly in the definition or instance to change its validation status.

> **Note:**
>
> One security privilege is required to upgrade an object's validation status to Quality Control, and a different privilege is required to upgrade an object's status to Production. If you have either of these privileges, you can set an object's status to Retired.

When an object meets the standards set by your organization for the next validation status, change the status as follows:

1. Navigate to the object definition or instance.
2. From the **Actions** drop-down list, select **Support Validation Info**. The Validation Status screen appears.
3. From the **Select Validation Status** drop-down list, select the correct status.
4. Click **Update**. The system tries to change the validation status and returns a message of success or failure.

   Validation Cascade. The system tries to upgrade related objects at the same time in accordance with the Validation Rules. If you are promoting an object instance to a higher validation status than its underlying definition, the system tries to promote the definition as well. If someone else has the definition checked out, the operation fails.

   If the definition contains secondary objects—such as Parameters— whose definitions are at a lower status, the system tries to promote them as well, and the operation fails if another user has checked them out.

Automatic Checkin. In most cases the system checks in the object and related objects as part of the validation process and applies the new status to the checked-in versions. However, you must manually check in an Execution Setup before you can change its validation status.

5. Click **Return**. The system returns you to the Properties screen for the object.

> **Note:**
>
> One security privilege is required to upgrade an object's validation status to Quality Control, and a different privilege is required to upgrade an object's status to Production. If you have either of these privileges, you can set an object's status to Retired.

Changing Outputs' Validation Status

To change an output's validation status, do the following:

1. Navigate to the output in one of these ways:
   - If you submitted the job, you can click the **Job ID** on the My Home page to reach the **Job Details** screen, and then click the **Output** to reach the **Output** screen.
   - In the Reports tab, navigate to the output according to its classifications and click its icon in the **Action** column.
2. From the **Actions** drop-down list, select **Support Validation Info** and click **Go**.
3. From the **Select Validation Status** drop-down list, select the correct status.
4. Click **Update**. The system changes the validation status of the output.
5. Click **Return**. The system returns you to the Properties screen for the output.

Adding Supporting Information

Before you promote an object to the next validation status, your organization may require that you provide documentation about the object: either a link to an output generated by the object, such as a report or log file, or a document such as a functional requirements document for the object.

- **Outputs**. You can use a log file to demonstrate that a Program or other executable ran successfully. You can use a report to demonstrate that an executable object has produced an appropriate report.
- **Documents**. You can upload any document as supporting information; for example, a requirements document or a set of test cases.

> **Note:**
>
> To add supporting information for an object definition, you must navigate to the definition in its Domain or Application Area. When you are in a Work Area you can add supporting information only for object instances.

This section contains the following topics.

- Adding a Supporting Document
  To link an object to a document as supporting validation information, do the following:
- Adding a Supporting Output
  To link an object to an output as supporting validation information, do the following:

Adding a Supporting Document

To link an object to a document as supporting validation information, do the following:

1. Navigate to the object.
2. From the **Actions** drop-down list, select **Support Validation Info**. The Validation Status screen appears.
3. Under Supporting Documents, click **Add**. The Manage Supporting Documents screen opens.
4. Enter a **Name** for the document.
5. Enter a **Description** for the document.
6. Click **Browse**. The system opens a standard Browse pop-up window.
7. Select any document on a local or shared drive and click **Open**.
8. Click **Apply**.

   The system returns to the Supporting Information screen with the new supporting output listed. The new supporting document's status is set to Active.

   Version History. If you upload a different document in the future, the system creates a new version of the supporting document. Click Version History to see who changed the document when, with the reason for change, which is required. You can also view the document uploaded for each previous version.
9. Click **Return** to go back to the object's Properties screen.

- Obsolete
  The system does not delete any supporting documents.

Obsolete

The system does not delete any supporting documents.

However, you can mark a document as Obsolete by selecting it and clicking **Obsolete**.

Adding a Supporting Output

To link an object to an output as supporting validation information, do the following:

1. Navigate to the object.
2. From the **Actions** drop-down list, select **Support Validation Info**. The Validation Status screen appears.
3. Under Supporting Outputs, click **Add**. The Manage Supporting Outputs screen opens.

4. Enter a **Description** for the output (required).
5. Click the **Search** icon to find the output you need. The Search and Select screen opens.
6. For **Search Using**, select either **Hierarchy** or **Job ID**.

   If you select **Hierarchy**, select the hierarchy name and the level to which the output is classified and enter the value in the **Search** field.

   Or enter a high-level value only and check **Include hierarchy based child in the search** to search all values in levels below the value you specify.

   If you select **Job ID**, enter the job ID in the **Search** field.
7. Click **Go**. The system displays the results of the search.
8. Click the icon in the **Quick Select** column for the output you want.

   The system returns to the Supporting Output Properties screen with the new output listed.
9. Click **Apply**. The system returns to the Managing Supporting Outputs screen with the new supporting output listed.
10. Click **Return**. The system returns to the Validation Status screen.
11. Click **Return**. The system returns to the object's Properties screen.

Validation Rules

The system enforces the following rules based on objects' validation status.

- You can change an object's validation status only when it is checked in. If the object is not checked in, the system tries to check it in. If another user has checked it out, you cannot change the validation status.
  Changing the validation status of a checked-in object does not change the version number of the object.
- You cannot promote an object instance to a validation status higher than its underlying object definition. However, the system tries to promote the underlying definition when you promote the instance. If the definition is checked out by another user, the operation fails.
  For example, if you create an instance of a Program definition whose validation status is Quality Control, you can promote the instance to a validation status of Quality Control, but you cannot promote it to Production until the definition has been promoted to Production. However, the system attempts to promote the definition to Production when you promote the instance, and if you have the necessary privileges on the definition, and if the definition is not checked out by a different user, the promotion succeeds.
- Validation status is version-specific. Each time you create a new version of an object by checking it out (including checking out a definition through an instance) the system automatically gives the new version a validation status of Development. The system does not change the validation status of any existing versions.
- When you copy an object, the validation status of the copy is set to Development no matter what the validation status of the original is.

- You cannot run an Execution Setup whose validation status is less than the validation status of its owning object instance unless you have the IQ Submit security privilege on the Execution Setup.
- To be installed in a Work Area, object instances must have a validation status equal to or greater than the Usage Intent of the Work Area.
- The Work Area cannot be promoted to a validation status higher than the validation status of any of its object instances. However, if you try to promote a Work Area to a status above that of any of its objects, you have the opportunity to promote all the object instances and their underlying definition versions to the same status in a cascade operation.
  The cascade validation fails if a different user has checked out any of the object definitions in need of promotion.
- No executables can be run in a Work Area until the Work Area's validation status is equal to or greater than its Usage Intent value, except by users with the special Install Qualify Submit privilege on the Work Area. This is to allow testing of the Work Area before making it generally available for use.
- Full installation and the Replace operation on Table instances in partial installation are not allowed in Work Areas with a usage intent of Production. This is to protect production data from deletion.
- You cannot remove a Table instance whose validation status is Production from a Work Area of any usage intent.

Rules for Retired Objects: The following rules apply to Retired objects:

- If an executable object instance is set to Retired, it cannot be submitted for execution, regardless of the validation status of it Execution Setups.
- If a Work Area is set to Retired, no executable objects within it can be submitted for execution.
- If you clone a Retired Work Area, the clone's status is also set to Retired. In the Work Area clone, the object instances retain the validation status they had in the original Work Area. Retired objects are included in the cloning process.

Reordering and Renumbering Objects

You can use the Reorder function to change the order of some objects and the Renumber function to create sequential numbering for some objects.

To reorder and/or renumber objects:

1. Click **Reorder**. The Reorder shuttle appears.
2. Select the object you want to move and click the **Up** and **Down** arrows to move it in relation to the other objects.
3. **Starting Entry Number** (Report Sets only). By default, the object displayed at the top is assigned the number one (1). You can enter a different starting number in the **Starting Entry Number** field.
4. If you want to change the object's numbers to reflect the new order and/or the new starting number, leave the **Renumber** flag set to Yes. If you change the setting to No, the system keeps the original numbers (except the changed starting number, if any) but displays the objects in the order you specify.
5. Click **Apply**. The new order and/or numbers are displayed in the original screens.

Defining and Mapping Table Descriptors

This section contains the following topics.

- About Table Descriptors
  Use Table Descriptors to map executable objects to the tables they read from and write to.
- Creating a Table Descriptor
  There are several ways to create a Table Descriptor.
- Mapping Table Descriptors to Table Instances
  A Program or other executable object instance can read data only when its source Table Descriptors are mapped to the Table instances that contain the source data, and can write data to Table instances only when target Table Descriptors are mapped to the Table instances.
- Mapping Columns
  When you map a Table Descriptor to a Table instance using any method, the system tries to map the Columns at the same time, by name. If the system cannot map the Columns, the mapping status of the Table Descriptor is set to Incomplete.
- Creating and Mapping Table Descriptors and Table Instances at the Same Time
  This section contains information on the following topics.
- Unmapping Table Descriptors
  If a Table Descriptor is mapped to one Table instance and you need to map it to a different Table instance, you must unmap it first.
- Modifying Table Descriptors
  To modify a Table Descriptor, you must also modify its source Table definition. You must check out both the Program, Load Set, Data Mart, or Business Area that contains the Table Descriptor and also check out the source Table definition. You can check out the Table definition from the Table Descriptor.

About Table Descriptors

Use Table Descriptors to map executable objects to the tables they read from and write to.

By having this table metadata in the executable object, you can map the executable to source and target table instances that are compatible in terms of column data type and length but that may not have exactly the same columns, table or column names, or column lengths as the tables the executable object was originally created to work with. This promotes reuse of Programs, Load Sets, Data Marts, and Business Areas.

In a Program definition's source code you refer to source and target Tables and Columns by the names of the Table Descriptors and their Columns, not the actual Table instances. In each instance of the Program you map the Table Descriptors to the Table instances they must read from or write to. You can have multiple instances of the same Program mapped to different source and target Table instances, and the Table instances may have different names or a different (but compatible) structure (see Mapping Columns of Different Data Types and Lengths).

For example, if Study 1 and Study 2 use demography tables with the same structure but called Demog in Study 1 and Demo in Study 2, you can create a single Program definition that reads from the demography table and creates several reports. You can

create one instance of the Program that reads from the Study 1 Demog Table and another instance of the Program that reads from the Study 2 Demo Table.

Table Descriptors do not have constraints or data processing types. Different rules apply to source and target Table Descriptors.

Source Table Descriptor: A source Table Descriptor must be mapped to a Table instance that contains data to be used as input. You can map a source Table Descriptor to a Table instance located in any Work Area to which you have security access; in other words, your Program (or Data Mart or Business Area) can operate or report on data located anywhere in Oracle LSH where you have View privileges to Table instances.

For example, if you have a different Application Area for each clinical trial, you can pull data from each trial and combine data for a whole project by mapping the source Table Descriptors of the Program that combines the data to Table instances in the Work Area for each of the clinical trials.

The same Table instance can serve as a source for multiple Programs, Business Areas, and Data Marts.

Target Table Descriptor: A target Table Descriptor must be mapped to a Table instance that receives data output from a Program or Load Set. Target Table Descriptors must be mapped to a Table instance located in the same Work Area as the Program or Load Set instance.

Mapping Rules: The system enforces the following mapping rules:

- Only one Program can write to a Table instance; therefore only one target Table Descriptor can be mapped to a particular Table instance.
- A Table instance can be mapped to only one Table Descriptor in a Program instance.
- For additional rules, see Using Format Conversions and Mapping Columns of Different Data Types and Lengths.

Creating a Table Descriptor

There are several ways to create a Table Descriptor.

See also:

- Setting Table Descriptor Attributes
- Mapping Table Descriptors to Table Instances
- Adding Source Table Descriptors
  This option is available for objects that have source Table Descriptors: Programs, Business Areas, and Data Marts.
- Adding Target Table Descriptors from a Remote Location
  This option is available for Oracle technology Programs and Load Sets.
- Adding Target Table Descriptors from a SAS File
  This option is available for objects that have target Table Descriptors and support this option: Programs and some Load Sets.
- Adding a Target Table Descriptor from an Existing Table Definition
  This option is available for objects that have target Table Descriptors: Programs and Load Sets.

- Adding a Table Descriptor from a Metadata File
  You can create aTable Descriptors by uploading a metadata (.mdd) file with a required format, or create multiple Table Descriptors at once by uploading a zipped file containing multiple metadata files.
- Adding a New Target Table Descriptor
  This option is available for objects that have target Table Descriptors: Programs and Load Sets.
- Setting Table Descriptor Attributes
  Table Descriptors may have the following attributes:
- Adding or Uploading Columns
  You can upload Columns based on SAS data set variables:

Adding Source Table Descriptors

This option is available for objects that have source Table Descriptors: Programs, Business Areas, and Data Marts.

To use this option, the Table instance you want to read from must already be defined in Oracle Life Sciences Data Hub and you must have Read privileges on it. You can create multiple Table Descriptors at a time from Table instances located in a single Work Area.

The system bases the Table Descriptor on the same Table definition the Table instance is based on, and automatically maps the new Table Descriptor to the Table instance.

> **Note:**
>
> This functionality is also available in the Actions drop-down as the **Table Descriptors from Existing Table Instances** item.

> **Note:**
>
> You can also create a source Table Descriptor by creating a target Table Descriptor and changing its Is Target attribute value to **No**.

1. In the Table Descriptors subtab in the Properties screen of a Program, Business Area, or Data Mart instance, click **Add Source**.

   The Create Table Descriptors from Table Instances screen opens.
2. Select the location of a Table instance for which you want to create a Table Descriptor:
   - Select the Table instance's Domain from the **Domain** drop-down list.
   - Select the Table instance's Application Area from the **Application Area** drop-down list. The choices are limited to Application Areas contained in the Domain you selected.
   - Select the Table instance's Work Area from the **Work Area** drop-down list. The choices are limited to Work Areas contained in the Application Area you selected.

3. Click **Go**. The system displays all the Table instances in the Work Area you selected.
4. Select one or more Table instances by clicking the **Select** checkbox.
5. Click **Create Table Descriptor**. The system returns you to the Create Table Descriptors from Table Instances screen. To return to the Program's Properties screen, click **Return**. Check that the mappings are complete.

Adding Target Table Descriptors from a Remote Location

This option is available for Oracle technology Programs and Load Sets.

The system searches for Variables in the same Application Area with the same name, data type, and length as each of the variables in the data set. If a matching Variable exists, the system bases a Column of the Table definition on it. If a Table definition with the same name already exists in the Application Area, the system appends _1 to it, or _*x* if the Table name already has a number appended, where *x* is the next larger integer.

1. After defining the Remote Location and any other required attributes, go to the Table Descriptors subtab of the Load Set's Properties screen and click Add Target from **Remote Location**. The system opens the Upload Table Descriptors screen with tables from the specified remote location displayed.
2. Select the tables from which you want to create Table Descriptors and click **Apply**.

   For each selected table, the system creates a target Table Descriptor and Table definition in the current Application Area. The system returns to the Load Set's Properties screen.
3. You can update the Table Descriptor as necessary; click the Table Descriptor name. See Setting Table Descriptor Attributes and Adding or Uploading Columns.
4. Click the Mapping icon for the Table Descriptor and map it to a Table instance. See Mapping Table Descriptors to Table Instances .

Adding Target Table Descriptors from a SAS File

This option is available for objects that have target Table Descriptors and support this option: Programs and some Load Sets.

If you specify a single data set file, the system creates a single Table Descriptor. If you specify a SAS transport file that contains more than one data set, the system creates one Table Descriptor for each data set in the file.

The system searches for Variables in the same Application Area with the same name, data type, and length as each of the variables in the data set. If a matching Variable exists, the system bases a Column of the Table definition on it. If a Table definition with the same name already exists in the Application Area, the system appends _1 to it, or _*x* if the Table name already has a number appended, where *x* is the next larger integer.

> **Note:**
>
> Oracle Life Sciences Data Hub gives SAS variables of SAS format BEST8 a length of 8 and Precision set to null.

1. In the Table Descriptors subtab of the Load Set or Program's Properties screen, click Add Target from **SAS File**. The system opens the Create Table Descriptors screen with **Create a New Table Definition and Descriptor from SAS file** selected.
2. Click **Browse**. The system opens a standard browse screen.
3. Navigate to the location of the SAS file.
4. Highlight the file and click **OK**, then click **Apply**

   The system creates a target Table Descriptor and Table definition in the current Work Area. The system returns to the Program or Load Set's Properties screen.
5. Click the Table Descriptor name to see the Table Descriptor screen. You can update the Table Descriptor here as necessary; see Setting Table Descriptor Attributes and Adding or Uploading Columns.
6. Click the Mapping icon for the Table Descriptor and map it to a Table instance. See Mapping Table Descriptors to Table Instances .

Adding a Target Table Descriptor from an Existing Table Definition

This option is available for objects that have target Table Descriptors: Programs and Load Sets.

1. In the Table Descriptors subtab of the Program or Load Set's Properties screen, click Add Target from **Library**. The system opens the Create Table Descriptors screen with **Create a Descriptor of an Existing Table Definition** selected.
2. Click the Search icon. The system opens the Search and Select screen.
3. Select the location of a Table definition from which you want to create a Table Descriptor:
   - Select the Table definition's Domain from the **Domain** drop-down list.
   - Select the Table definition's Application Area from the **Application Area** drop-down list. The choices are limited to Application Areas contained in the Domain you selected. Leave this field blank to search for definitions contained in the Domain library.
   - If you know the exact name or version label of the definition you want, select either Name or Version Label from the **Search By** drop-down list and enter the name or version label in the blank field.
   - (Optional) Select **Display All Versions** and/or **Display Not Null Version Labels**.
4. Click **Go**. The system displays the Table definitions in the Application Area or Domain Library you selected.
5. Click the Quick Select icon for the Table definition from which you want to create the Table Descriptor. The system populates the **Definition Source** field with the definition you selected.
6. Click **Apply**. The system opens the screen for the new Table Descriptor.
7. You can update the Table Descriptor here as necessary; see Setting Table Descriptor Attributes and Adding or Uploading Columns.
8. Click the Mapping icon for the Table Descriptor and map it to a Table instance. See Mapping Table Descriptors to Table Instances .

Adding a Table Descriptor from a Metadata File

You can create aTable Descriptors by uploading a metadata (.mdd) file with a required format, or create multiple Table Descriptors at once by uploading a zipped file containing multiple metadata files.

Depending on the type of object for which you are creating the Table Descriptor, the name of the button in the Table Descriptor tab varies: either Add, or Add Target, or File.

Creating a Metadata File: If the file has an .mdd extension, the system expects a set of Column attribute values, optionally preceded by a row identifying the delimiter and a row defining Table attribute values, each of which must begin with a key word. A row beginning with dashes is treated as a comment. For example:

```
--This is a comment.

Example 3-1: Sample Set Metadata File
lsh_delimiter = ,
--This section is for the Table attributes
lsh_table= DM,DEMOG Table,EMP,EMP,EMP,Staging with
Audit,Yes,Yes,Blinded,target,yes,yes

--The following are columnsComment Line 2
INITS,VARCHAR2,100,,inits,inits,2.,,inits,no,1
AGE,NUMBER,10,,2,age,age,2.,,age,yes,1
    DOB,DATE,,,3,dob,dob,datetime.,,dob,yes,1,MM/DD/YY HH24:MI:SS
```

Delimiter: The first row defines the delimiter used in the file. If not specified, Oracle LSH treats it as a comma delimited file. The delimiter row must begin with `lsh_delimiter=`

Table and Table Descriptor Attributes: The second row lists the table attributes required in the file. The Table attribute row must begin with `lsh_table=`

If the second row is not present or contains null values, the system assumes that the filename (without extension) is the Table Name and follows the normal Oracle LSH default behavior for the attribute values. The attributes and their required order in the file are: `Name, Description, Oracle Name, SAS Name, SAS Label, Process Type, Allow Snapshot?, Blinding Flag?, Blinding Status, SAS Library Name, Is Target?, Target as Dataset?`

Some attributes have associated reference codelists and allow either the actual values for the associated reference codelist (RC) columns or the decode values defined in the "Meaning" attribute of the _RC lookup. For example, "select meaning from cdr_lookups where lookup_code='< RC>' so that **YES** or **Yes** or **$YESNO$YES** are acceptable values.

The following table outlines the applicable values for each Oracle LSH Table attribute that has an associated reference codelist.

> **Note:**
>
> Processing types that require audit keys are not supported.

Table 3-1 Table Attributes with Reference Codelist Values

Table Attribute	Applicable Reference Codelist Values
Processing Type	Staging with Audit, Staging without Audit, Transactional High Throughput, Transactional without Audit
Allow Snapshot	Yes, No
Blinding Flag	Yes, No
Blinding Status	Blinded, Unblinded
Is Target	Yes, No
Target as dataset	Yes, No
Data Type	Date, Number, Varchar2
Nullable	Nullable , Yes, No

Columns: Subsequent rows must contain the column and variable attributes with each represented by a new row in the text file with attributes. The position is determined by the order in which the column /variable rows are processed. For example, in a comma delimited file: `Name, Data Type, Length, Precision, Oracle Name, SAS Name, SAS Format, Description, SAS Label, Nullable, Default Value, Date Format`

Normal Oracle Life Sciences Data Hub validation rules apply to the Column or variable attributes. The operation uses Oracle LSH default values if invalid values are provided for any of the attributes.

Adding a New Target Table Descriptor

This option is available for objects that have target Table Descriptors: Programs and Load Sets.

If your Program will write to (or read from) a table that does not yet exist, you can manually create the Table definition and Table Descriptor at the same time:

1. In the Table Descriptors subtab of the Program's Properties screen, click Add Target from **New**. The system opens the Create Table Descriptors screen with **Create a New Table Definition and Descriptor** selected.
2. Enter values in the following fields:
   - **Name**. See Naming Objects
   - **Description**. See Creating and Using Object Descriptions.
   - **Oracle Name** (up to 30 characters, uppercase, no spaces). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces, truncated to 30 characters if necessary.

> **Note:**
>
> Each Table Descriptor within a particular executable object must have a unique Oracle Name.

- **SAS Name** (up to 32 characters, uppercase, no spaces). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces.
- Enter a **SAS Label** (optional, up to 256 characters). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field.
- Enter a **SAS Library Name** (optional, up to 8 characters).

> **Note:**
>
> If you plan to use this Table Descriptor with an Oracle Business Intelligence (OBIEE) Business Area, you can set the SAS Library Name to `$REPINIT`. See Defining Table Descriptors for more information.

3. Set Is Target and other Table Descriptor attributes; see Setting Table Descriptor Attributes.
4. In the **Classification** section, select the following for the definition:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
5. Click **Apply**. The system opens the screen for the new Table Descriptor.

   You can update it as necessary:
   - Click **Update** to modify attribute settings. See Setting Table Descriptor Attributes.
   - Check out the Table definition to be able to add Columns. See Adding or Uploading Columns.
6. If you make any changes, click **Apply** to save them. The system opens the Table Descriptor Properties screen. Click **Return**. The system opens the Program's Properties screen.
7. Click the Mapping icon for the Table Descriptor and map it to a Table instance. See Mapping Table Descriptors to Table Instances .

Setting Table Descriptor Attributes

Table Descriptors may have the following attributes:

Is Target: Select **Yes** to create a target Table Descriptor. Select **No** to create a source Table Descriptor.

Target As Dataset: (Available only in SAS Programs, and only when **Is Target** is set to **Yes**.) Select **Yes** if you are using a legacy SAS program that uses data statements to write to data sets. Because Program source code must write to Table Descriptors, and Table Descriptors are views, you should use Proc SQL statements to write data to tables in Oracle Life Sciences Data Hub. However, if you set this attribute to Yes, Oracle Life Sciences Data Hub adds a processing step to enable SAS data statements to write to Oracle Life Sciences Data Hub Table Descriptors. This extra processing step results in slower performance but allows you to use existing programs.

Select **No** if the Program's source code uses Proc SQL statements to write to Table Descriptors. This results in optimal performance.

Reveal Audit: (Available only for source Table Descriptors; when **Is Target** is set to **No**.) By default **Reveal Audit** is set to **No** and the Program, Data Mart, or Load Set can read data from the Table instance mapped to this Table Descriptor that is or was current at a particular point in time. By default it sees only current data, but it is possible to specify a data snapshot at a past point in time in the Execution Setup of the Program, Data Mart, or Load Set.

Each time a data record is changed in Oracle Life Sciences Data Hub, the system creates a new row with the updated information. The system sets the end timestamp of the old record so that it is no longer current. When a record is deleted, the system sets its end timestamp and also adds a row explicitly recording the deletion. The set of rows for a single data record constitutes its audit trail.

If you set **Reveal Audit** to **Yes** the system exposes all data in the Table instance at all points in time to the Program, Data Mart, or Load Set reading from the Table instance, and exposes the predefined audit columns CDR$CREATION_TS (creation timestamp), CDR$CREATED_BY (creator username), CDR$MODIFICATION_TS (modification timestamp), and CDR$MODIFIED_BY (modifier username).

> **Note:**
>
> Because SAS does not support the $ special character, the SAS Name of each of these internal column names has an underscore (_) instead of a dollar sign ($) as follows: CDR_CREATION_TS, CDR_CREATED_BY, CDR_MODIFICATION_TS, CDR_MODIFIED_BY.

You can use this functionality as follows:

- **Data Marts**. If you set **Reveal Audit** to **Yes** for all the Table Descriptors in a Data Mart, the resulting Data Mart output contains the complete audit trail of every record that the mapped Table instances ever contained.
- **Programs**. If you set **Reveal Audit** to **Yes** for all the Table Descriptors in a Program, you can write source code that reads data from all points in time and that references the audit columns.
- **Load Sets**. Because the source Table Descriptors of most types of Load Sets point to data files or tables outside of Oracle LSH that do not contain audit information, it does not make sense to use the Reveal Audit feature with SAS, text, or most Oracle Load Sets. However, Oracle Clinical uses a similar system for maintaining an audit trail so you may want to use Reveal Audit with some Oracle Clinical tables.
  Oracle Clinical inserts a row each time a record is modified and sets the end timestamp of the previous row for the same record. However, when a record is deleted Oracle Clinical sets its end timestamp but does not add a row explicitly recording the deletion.

- **Note:**

  The Execution Setup user interface cannot detect if a Table instance is mapped to a Table Descriptor whose **Reveal Audit** flag is set to **Yes**, so it is possible for a user to specify a snapshot for such a Table instance at execution time. However, during execution the system ignores any snapshots set for Table instances mapped to a source Table Descriptor whose **Reveal Audit** flag is set to **Yes**.

Adding or Uploading Columns

You can upload Columns based on SAS data set variables:

1. Check out the Table Definition if it is not already checked out.
2. Click **Upload Column**. The system displays the Upload Column screen.
3. Click **Browse**. The system opens a standard Browse pop-up window.
4. Navigate to the data set from which you want to upload Columns and click **Open**, then **Apply**. The system uploads all the variables from the data set as Columns in the Table Descriptor and its source Table definition.

   You can remove any Columns you do not need. To remove one or more Columns, click the checkbox in the Select column and click **Remove**.

Mapping Table Descriptors to Table Instances

A Program or other executable object instance can read data only when its source Table Descriptors are mapped to the Table instances that contain the source data, and can write data to Table instances only when target Table Descriptors are mapped to the Table instances.

In addition, both the executable instance and the Table instance must be installed in the database. Table Descriptors must be mapped to Table instances at both the Table and Column level.

Mapping Columns: When you map a Table Descriptor to a Table instance, the system may or may not be able to map Columns automatically. If not, you can map them manually; see Mapping Table Descriptors Manually.

However, if a Target Table Descriptor in a Load Set or Program is updated or new columns added, installation automatically upgrades the Table instance and completes the mapping. Note that the Table instance is upgraded only if the changes to the Table Descriptor are non-destructive, such as the addition of a column.

You can supply default values for any Columns that are still unmapped; see Mapping Columns to Constants. In some cases, you can map columns of different data types and lengths; see Mapping Columns of Different Data Types and Lengths).

For source Table Descriptors you do not need to map Table instance Columns that are not used by the Program.

Data Processing Compatibility: Be sure to map target Table Descriptors to Table instances whose data processing type is compatible with your source code:

- If you have insert, update, and delete statements in your source code, you must map your target Table Descriptors to Table instances with a data processing type of either Transactional or Staging.
- If you have insert statements only, you must map your target Table Descriptors to Table instances with a data processing type of either Reload or Staging.

This section includes the following topics:

- Automatic Mapping by Name
  The Automatic Mapping by Name job can run on multiple Table Descriptors in a Program or other executable object. By default, it looks for a Table instance with the same name in the same Work Area as the executable instance.
- Mapping Table Descriptors Manually
  If you want to map a Table Descriptor to a Table instance that has a different name, you must map it manually. You can also use this method to remap a Table Descriptor that is already mapped.
- Remapping
  You can remap a mapped Table Descriptor to a different Table instance in two ways.
- The Effects of Modifying the Table Descriptor or Table Instance on a Mapping
  A mapping is specific to a particular version of both the Table Descriptor and the Table instance.

Automatic Mapping by Name

The Automatic Mapping by Name job can run on multiple Table Descriptors in a Program or other executable object. By default, it looks for a Table instance with the same name in the same Work Area as the executable instance.

If you are mapping source Table Descriptors only, you can specify a different Work Area.

You can also use this feature for remapping Table Descriptors.

To run automatic mapping:

1. On the executable instance's Properties screen, select **Automatic Mapping By Name** from the **Actions** drop-down list and click **Go**. (In a Report Set instance, you can reach this screen from the Report Set structure screen by selecting the Report Set instance or a Report Set Entry, then selecting **Map** from the drop-down list and clicking **Go**.)

   The system displays the Automatic Mapping by Name screen and, by default, unmapped Table Descriptors only. The current Work Area is selected by default, and if it contains a Table instance with the same name as an unmapped Table Descriptor, the Table instance is displayed on the same row as the Table Descriptor.
2. In the **View** drop-down list, specify which Table Descriptors to display:
   - **Unmapped**. The system displays only unmapped Table Descriptors (the default).
   - **Mapped**. The system displays only mapped Table Descriptors, so that you can remap them to different Table instances.

- **Both Mapped and Unmapped**. The system displays all Table Descriptors and you can map and remap them as necessary.

3. Select the location of a Table instance you want to map to. Accept the current Work Area or, only if you are mapping a source Table Descriptor, specify a different one by doing the following:
   - Select the Table instance's Domain from the **Domain** drop-down list.
   - Select the Table instance's Application Area from the **Application Area** drop-down list. The choices are limited to Application Areas contained in the Domain you selected.
   - Select the Table instance's Work Area from the **Work Area** drop-down list.
   - Click **Go**. The system searches the selected Work Area for Table instances with a name that matches the name of any of the displayed Table Descriptors and displays them in the same row as the matching Table Descriptor if found.
4. Select one or more Table Descriptors with a matching Table instance in the specified location and click **Map**. The system creates the Table-level mapping and attempts to map Columns by name.
5. Map another Table Descriptor or click **Return** to display the executable's main page.
6. Check the mapping status of the Table Descriptors you have mapped.
   - If the system was able to map all their Columns, their mapping status is Complete.
   - If the system could not map all the required Columns, their mapping status is Incomplete and you must map them manually. See Mapping Columns.

Mapping Table Descriptors Manually

If you want to map a Table Descriptor to a Table instance that has a different name, you must map it manually. You can also use this method to remap a Table Descriptor that is already mapped.

To map a Table Descriptor to a Table instance manually, go to the main Program screen, Table Descriptors subtab, and do the following:

1. Click the icon in the Table Descriptor's **Mapping** column. The system opens the Mapping screen with the Table Descriptor's name already entered in either the From or To column, depending on whether it is a target or a source Table Descriptor, respectively.
2. Click **Update**. Several fields become modifiable.
3. Click the **Search** icon next to the **Map to Table Instance** field. The system opens the Search screen.
4. Select the **Domain**, **Application Area**, and **Work Area** of the Table instance to which you want to map.
   - Select the Table instance's Domain from the **Domain** drop-down list.
   - Select the Table instance's Application Area from the **Application Area** drop-down list. The choices are limited to Application Areas contained in the Domain you selected.

- If you know the exact name or version label of the Table instance you want, select either Name or Version Label from the **Search By** drop-down list and enter the name or version label in the blank field.

5. Click **Go**. The system displays all the Table instances that meet the criteria, and for which you have View privileges.

   You cannot map to a Table instance while its source Table definition is checked out. The system displays Table instances whose source Table definition is currently checked out as grayed out. You can look at the Table definition to see who has it checked out.

6. Click the **Quick Select** icon for the Table instance you want to map.

   The system enters the name of the Table instance you selected and automatically tries to map it to the Table Descriptor. The system displays any Columns it finds that match a Column in the Table Descriptor across from the matching Column. If the system does not find a match, it populates a drop-down list for each Table instance Column that contains the names of all the Table instance Columns.

7. For each Column of the Table instance, choose the Column from the drop-down list that you want to map to the Table Descriptor Column displayed on the same row.

   Alternatively, you can map to a constant (see Mapping Columns to Constants) or convert to a different data type (see Mapping Columns of Different Data Types and Lengths) you can enter that information in the **Default Value** or **Format String** field, respectively.

8. Click **Apply**. The system saves the mapping and returns you to the Program screen.

Remapping

You can remap a mapped Table Descriptor to a different Table instance in two ways.

- If the Table Descriptor has the same name as the Table instance, use automatic mapping by name; see Automatic Mapping by Name.
- If the Table Descriptor has a different name from the Table instance, use manual mapping; see Mapping Table Descriptors Manually.

The Effects of Modifying the Table Descriptor or Table Instance on a Mapping

A mapping is specific to a particular version of both the Table Descriptor and the Table instance.

When you check out either one:

- The system upgrades the mapping to the new version of the Table Descriptor or Table instance if possible. There are two conditions where it is not possible: You modify a Table Descriptor from a source to a target and either:
  - The Table instance is not in the same Work Area as the executable object that owns the Table Descriptor.
  - The Table instance is already mapped to another target Table Descriptor.
- The system reevaluates the mapping and resets its status to either Complete or Incomplete.

For a Mapping to be complete, all Columns must be mapped to Columns in the Table instance or to constants, and all other mapping rules must be followed.

When you check out either a mapped Table Descriptor or a mapped Table instance, there is no effect on the other object, only on the mapping.

Mapping Columns

When you map a Table Descriptor to a Table instance using any method, the system tries to map the Columns at the same time, by name. If the system cannot map the Columns, the mapping status of the Table Descriptor is set to Incomplete.

You must map the Columns manually before you can install the Program instance.

In most cases you map each Column of the Table Descriptor to a Column in the Table instance to which it is mapped. However, you can also map Columns to constants.

This section includes the following topics:

- Mapping Columns Automatically
  If you have mapped a Table Descriptor to a Table instance, but have not mapped the columns, then the system can map columns automatically.
- Mapping Columns Manually
- Mapping Columns to Constants
  There may be cases where a Table instance or Table Descriptor upstream in the data flow does not contain a Column needed for mapping by a Table Descriptor or Table instance Column downstream in the data flow.

Mapping Columns Automatically

If you have mapped a Table Descriptor to a Table instance, but have not mapped the columns, then the system can map columns automatically.

Do the following:

1. In the Program instance's Properties screen, Table Descriptors subtab, click the icon in the **Mapping** column for the Table Descriptor whose Columns you want to map.
2. Click the **Reset to Default Mapping** button. For each Table Descriptor, the system checks the Table instance for a column with a matching name or a matching variable reference. If it finds one, it maps the Table Descriptor column with the Table instance column.

Nothing happens if you click this button when the Table Descriptor and Table instance mapping is already complete.

If the system cannot find matching column names or variable reference names between the Table Descriptor and the Table instance, then you cannot automatically map in this way. You need to map columns manually as described in the next section.

Mapping Columns Manually

To map Table Descriptor Columns to Table instance Columns, do the following:

1. In the Program instance's Properties screen, Table Descriptors subtab, click the icon in the **Mapping** column for the Table Descriptor whose Columns you want to map.

   The Mapping screen opens. It has a row for each Column divided into a set of columns labeled From and another set labeled To. The system populates the From and To sections as appropriate for the flow of data: if the Table Descriptor is a source Table Descriptor, it is displayed in the To section because data appears to flow from the Table instance to the Program through the Table Descriptor; if it is a target Table Descriptor, it is displayed in the From portion because data appears to flow through it to the target Table instance. (Table Descriptors never actually contain data.)

2. Click **Update**. The system makes several fields enterable.

3. **Select a Table instance**. If a Table instance is not already specified in the upper portion of the screen, or if you want to map the Table Descriptor to a different Table instance, click the Search icon by the Map to Table Instance field.

   The Search and Select screen opens with the current Work Area's Table instances displayed by default.

   If the Table Descriptor is a target Table Descriptor, you cannot select a different Work Area. If the Table Descriptor is a source Table Descriptor, you can select a Table instance in any Work Area:

   - Select the Table instance's Domain from the **Domain** drop-down list.
   - Select the Table instance's Application Area from the **Application Area** drop-down list. The choices are limited to Application Areas contained in the Domain you selected.
   - If you know the exact name or version label of the Table instance you want, select either Name or Version Label from the **Search By** drop-down list and enter the name or version label in the blank field.

   Click the Quick Select icon for the Table instance you want to map to. the system returns you to the Mapping screen with the Table instance information displayed.

4. **Map Columns**. Map each Table Descriptor Column to a Table instance Column or a default value. You can also enter a conversion string if either the Table instance Column or the constant is in a different format from the Table Descriptor Column:

   - Select a Table instance **Column** from the drop-down list to which to map the Table Descriptor Column.
   - Enter a value in the **Default Value** field to populate the Column to a constant; see Mapping Columns to Constants.
   - If you are mapping Columns of different data types, enter a format string for the system to use in the conversion, if necessary. See Using Format Conversions.

5. Click **Apply**. The system validates the mappings you specified and returns you to the Program instance's Properties screen.

   If you specified an invalid mapping, the system displays an error message.

- Using Format Conversions
  If you map to a Table instance Column or a constant with a different data type or format than the Table Descriptor's Column, you can enter a format conversion string for Oracle LSH to convert the value that is upstream in the data flow to the correct format for the downstream value.

- Mapping Columns of Different Data Types and Lengths
  If the data types and/or lengths of the Table instance and Table Descriptor Columns do not match, mapping may still be allowed. The rules are different for source and target Table Descriptors.

Using Format Conversions

If you map to a Table instance Column or a constant with a different data type or format than the Table Descriptor's Column, you can enter a format conversion string for Oracle LSH to convert the value that is upstream in the data flow to the correct format for the downstream value.

For example, if a source Table instance contains a number value that needs to be converted to a dollar amount for the Program to analyze, enter a format conversion such as: `format=$99,999.00`. If a source Column has a data value of 10000, Oracle LSH converts that value to $10,000.00.

For format models, see the *Oracle 9i SQL Reference Guide*.

Mapping Columns of Different Data Types and Lengths

If the data types and/or lengths of the Table instance and Table Descriptor Columns do not match, mapping may still be allowed. The rules are different for source and target Table Descriptors.

See Figure 3-3 for an illustration of the data flow.

The principles are:

- Only Number data types can feed into Number data types and only Date data types can feed into Date data types.
- Shorter lengths can feed into longer lengths, but longer lengths cannot feed into shorter lengths.
- For Number data types, lesser precision can feed into greater precision, but greater precision cannot feed into lesser precision.
- On the source side, Table instance Columns of Number and Date data types can feed into Table Descriptor Column Varchar2 data types.
- On the target side, Table Descriptor Column Number data types can feed into Table instance Column Varchar2 data types, but Date data types cannot.

Mapping Columns to Constants

There may be cases where a Table instance or Table Descriptor upstream in the data flow does not contain a Column needed for mapping by a Table Descriptor or Table instance Column downstream in the data flow.

- A source Table instance does not contain a Column that is needed for mapping by a source Table Descriptor
- A target Table Descriptor does not have a Column needed for mapping by the target Table instance

In these cases, you can map the existing Column of the source Table Descriptor or target Table instance to a constant by supplying a default value for the Column. Figure 3-3 shows how to supply constants for Columns in relation to the data flow direction.

If you are supplying a constant as a value to a Column whose data type is not Varchar2, you must provide conversion information. For example, to convert the text you enter as the constant to the Date data type, enter the value 04-JAN-2005 in the **Default Value** column, and enter DD-MON-YYYY in the **Format String** column. The system reads the constant as a date.

For any data type, the value after conversion cannot exceed the length defined for the Column.

NULL is a valid constant value (if allowed by the receiving Column).

When you pass a constant to a Column downstream from it in the data flow (either to a source Table Descriptor Column or a target Table instance Column (see Figure 3-3), the specific requirements differ according to the data type of the receiving Column, as follows:

Varchar2: If you pass a constant to a Column with a data type of Varchar2, the length of the constant must be less than or equal to the length of the Column.

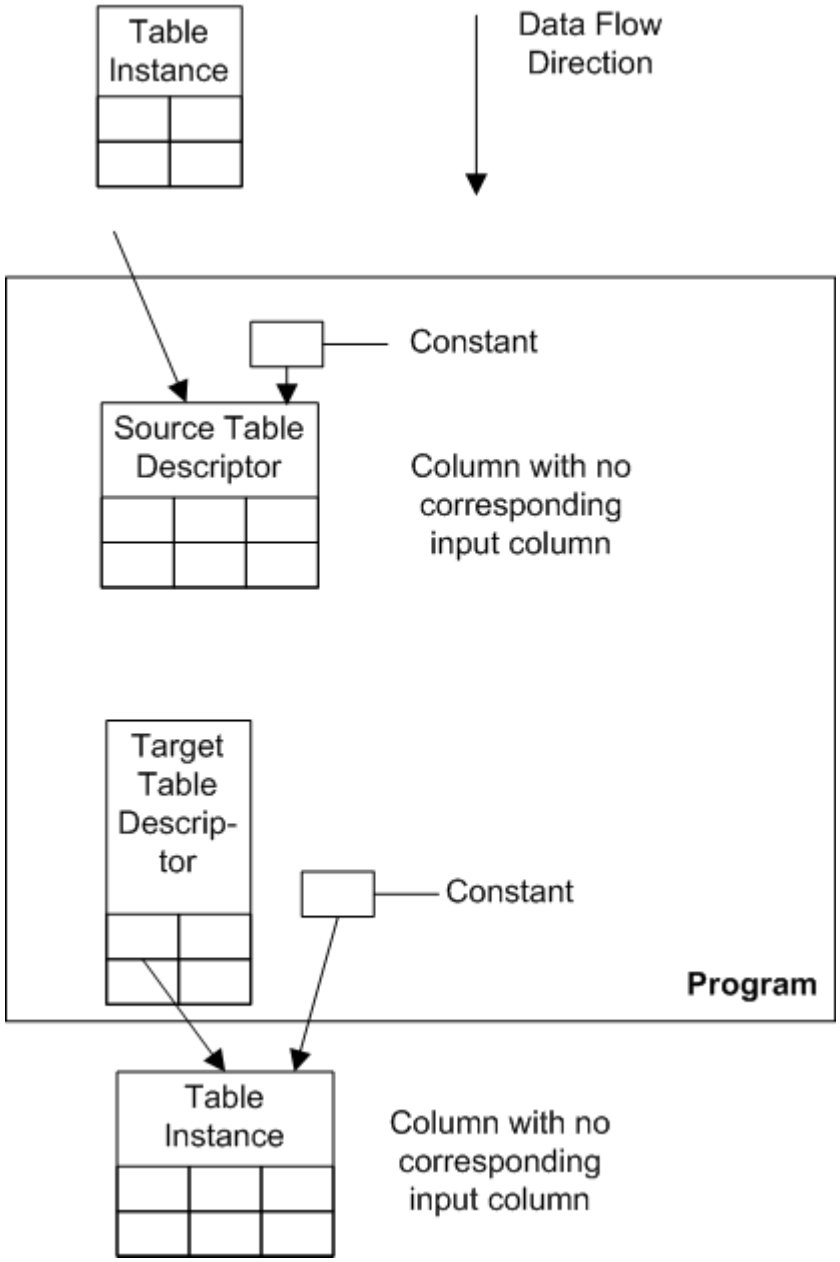
Number: If you pass a constant to a Column with a data type of Number, the following rules apply:

- You must provide a number format (with a default) and the constant must be valid when passed to a to_number with the supplied number format.
- The length of number produced by to_number (constant, number format) must be <= the length of the Column.
- The precision of number produced by to_number (constant, number format) must be <= the precision of the Column.

Date: You must provide a date format and the constant must be valid when passed to a to_date with the supplied number format.

Null: Allowed EXCEPT if mapping from a target Table Descriptor to a Table instance Column that is not nullable.

Figure 3-3 Data Flow from Table Instances to Program Instance to Table Instances



Creating and Mapping Table Descriptors and Table Instances at the Same Time

This section contains information on the following topics.

- Creating and Mapping Table Instances during Single Instance Installation
  You can install an object from its Properties screen or Work Area even if its installation status is Non Installable. For objects that have target Table

Descriptors, such as Programs and Load Sets, the system attempts to correct issues that are preventing the object from being installed:

- Creating Table Descriptors from Table Instances and Simultaneously Mapping Them
  If the Table instance you need to read from or write to already exists in Oracle LSH, you can create a Table Descriptor from the Table instance. The system bases the Table Descriptor on the same Table definition the Table instance is based on, and automatically maps the new Table Descriptor to the Table instance.
- Creating Table Instances from Table Descriptors and Simultaneously Mapping Them
  You can create a Table instance in the current Work Area corresponding to each Table Descriptor and map each pair. The system bases the Table instance on the same Table definition the Table Descriptor is based on, and automatically maps the Table Descriptor to the new Table instance.

Creating and Mapping Table Instances during Single Instance Installation

You can install an object from its Properties screen or Work Area even if its installation status is Non Installable. For objects that have target Table Descriptors, such as Programs and Load Sets, the system attempts to correct issues that are preventing the object from being installed:

1. The system runs Automatic Mapping by Name to map any unmapped target Table Descriptors to Table instances with the same name in the current Work Area, if any.
2. If any target Table Descriptors remain unmapped, the system automatically creates new Table instances in the current Work Area and maps them.

Both these tasks are performed by a batch job. If the system cannot resolve all issues automatically, the job fails. You can view the job log in your My Home screen.

Creating Table Descriptors from Table Instances and Simultaneously Mapping Them

If the Table instance you need to read from or write to already exists in Oracle LSH, you can create a Table Descriptor from the Table instance. The system bases the Table Descriptor on the same Table definition the Table instance is based on, and automatically maps the new Table Descriptor to the Table instance.

You can use this method to create multiple Table Descriptors at the same time from Table instances located in a single Work Area.

This is the only method where the system creates a source Table Descriptor by default. You can change the setting to Is Target if necessary.

> **Note:**
>
> You can create target Table Descriptors only from Table instances in the same Work Area as the Program.

To create a Table Descriptor from an existing Oracle LSH Table instance, do the following:

1. In the Program's Properties screen, select **Table Descriptors from Existing Table Instances** from the **Actions** drop-down list and click **Go**.

   The Create Table Descriptors from Table Instances screen opens.

2. Select the location of a Table instance for which you want to create a Table Descriptor:
   - Select the Table instance's Domain from the **Domain** drop-down list.
   - Select the Table instance's Application Area from the **Application Area** drop-down list. The choices are limited to Application Areas contained in the Domain you selected.
   - Select the Table instance's Work Area from the **Work Area** drop-down list. The choices are limited to Work Areas contained in the Application Area you selected.
3. Click **Go**. The system displays all the Table instances in the Work Area you selected.

> **Note:**
>
> A Table instance can be mapped to only one Table Descriptor in a Program instance. In addition, only one Program instance can write to a particular Table instance. You get an error if you select a Table instance whose selection would violate either of those rules.

4. Select one or more Table instances by clicking the **Select** checkbox.
5. Click **Create Table Descriptor**. The system returns you to the Create Table Descriptors from Table Instances screen. To return to the Program's Properties screen, click **Return**. Check that the mappings are complete.

Creating Table Instances from Table Descriptors and Simultaneously Mapping Them

You can create a Table instance in the current Work Area corresponding to each Table Descriptor and map each pair. The system bases the Table instance on the same Table definition the Table Descriptor is based on, and automatically maps the Table Descriptor to the new Table instance.

You can use this method to create multiple Table instances at the same time from Table Descriptors located in a single Load Set, Program, Data Mart, Business Area, Report Set or Workflow. In Report Sets and Workflows this option is available only at the Report Set and Workflow level, not from individual Program instances in the Report Set or Workflow.

> **Note:**
>
> Table instances are created with the default settings for all attributes, including Process Type (which is Staging with Audit). You can go to the Table instances and update them as necessary.

To create a Table instance from one or more existing Table Descriptors in a single Load Set, Program, Data Mart, or Business Area, do the following:

1. In the object's Properties screen, select **Table Instances from Existing Table Descriptors** from the **Actions** drop-down list and click **Go**.

   The Create Table Instances from Table Descriptors screen opens, displaying all the unmapped Table Descriptors.

   > **Note:**
   >
   > If you have created the object inside a Workflow, then use the Actions drop-down list job **Table Instances from Existing Tables Descriptors** from the Workflow's Properties screen. This job is not available in the object's Properties screen.

2. Select the Table Descriptors for which you want to create a Table instance by clicking the box in the **Select** column.
3. Click **Create Table Instance**. The system displays a message asking if you want to create the Table instance.
4. Click **Yes**.

   The system returns you to the Create Table Instances from Table Descriptors screen. To return to the Program's Properties screen, click **Return**. Check that the mappings are complete.

Unmapping Table Descriptors

If a Table Descriptor is mapped to one Table instance and you need to map it to a different Table instance, you must unmap it first.

To unmap a Table Descriptor, do the following:

1. In an executable object instance, click the Table Descriptor you want to unmap.
2. Click the icon in the Mapping column. The Mapping page opens.
3. Click **Update**. The screen becomes editable and the **Unmap** button appears.
4. Click **Unmap**. The system unmaps the Table Descriptor.

You can then click the **Reset to Default Mapping** button to automatically map columns according to column names, or remap the Table Descriptor using any of the methods described in this section; see Mapping Table Descriptors to Table Instances .

Modifying Table Descriptors

To modify a Table Descriptor, you must also modify its source Table definition. You must check out both the Program, Load Set, Data Mart, or Business Area that contains the Table Descriptor and also check out the source Table definition. You can check out the Table definition from the Table Descriptor.

Any Table instances and Table Descriptors that are based on the original version of the source Table definition continue to reference the same version unless you choose to upgrade them to the new version; see Upgrading Object Instances to a New Definition Version.

You can also remove a Table Descriptor and add a different one.

Creating, Modifying, and Submitting Execution Setups

This section contains the following topics.

- About Execution Setups and Templates
  In order to run an Oracle LSH executable object—Program, Load Set, Report Set, Workflow, or Data Mart—you must create at least one Execution Setup for it.
- Creating an Execution Setup
  The system creates an Execution Setup automatically if you click **Submit** directly in the Properties screen of an executable object instance and no Execution Setup currently exists.
- Modifying an Execution Setup and Setting Parameters
  After you have generated a default Execution Setup you can modify the system Parameters and the default values of the runtime Parameters. The Parameters in the Execution Setup are copies of those in the executable object definition. Modifying them has no effect on the object definition.
- Report Set and Workflow Execution Setups
  For information on submitting a Report Set or Workflow for execution, see "Running a Job from Reports, My Home, or Applications" in the *Oracle Life Sciences Data Hub User's Guide*.
- Removing an Execution Setup
  You can remove an Execution Setup if you have delete privileges on Execution Setups of the relevant subtype and also Modify privileges on the parent object's subtype.
- Activating a Version of an Execution Setup
  Only one version of an Execution Setup can appear in the Reports tab for submission at any one time.
- Submitting an Execution Setup
  When you have created an Execution Setup, you can submit it to execute the Program, Load Set, Report Set, Workflow, or Data Mart, by clicking **Submit** in the Execution Setups screen or in the Reports tab.

About Execution Setups and Templates

In order to run an Oracle LSH executable object—Program, Load Set, Report Set, Workflow, or Data Mart—you must create at least one Execution Setup for it.

Execution Setups become the submission form for the executable object. They can be displayed in the Reports tab so that users with the appropriate privileges can submit the associated Program, Load Set, Report Set, Workflow, or Data Mart for execution.

You can also create Execution Setups to submit a job at a single future point in time, on a regularly scheduled basis, or when triggered by the receipt of an XML message from an external system. These Execution Setups are not displayed in the Reports page, but run automatically.

Execution Setups include a predefined set of system Parameters; see System Parameters. They also include all the user-defined input and input/output Parameters defined as visible in the executable object. These Parameters are copies of the Parameters in the object and belong only to the Execution Setup. You can modify them without affecting the executable object itself. You determine whether these Parameters are visible to and settable by the person submitting the job.

More than one person can work on an Execution Setup at the same time, if they work on different Parameter Sets. For example, one person can work on a Program's Parameter settings and another person can work on system Parameters in the same Execution Setup at the same time.

You can define multiple Execution Setups for the same executable object instance. For example, you could define one Execution Setup for immediate execution, to be used to test the Program, and another Execution Setup to run at regular intervals, for use on production data. You could also define multiple Execution Setups with their Parameters bound to different values.

Copying Execution Setups: You can create a copy of one or more Execution Setups for use with the same instance and then modify them as necessary.

Execution Templates: If you create an Execution Setup that you think would be useful to other instances of the same executable definition, you can make it available for use as an Execution Template.

When you create an Execution Setup, you have the option of creating it from an Execution Template. If you choose this option, the system displays a list of all the Execution Setups that have been designated as Execution Templates in all instances of the same object definition version, if any. You can select one and modify it as necessary.

Forward Chaining: Forward chaining is a special type of processing that executes the programs automatically that directly or indirectly uses the data in a table, whenever that table is updated. A single job—for example, the execution of a Load Set to load data into Oracle LSH—triggers jobs that read from the tables the original job writes to, and then triggers the jobs that read from tables the second set of programs writes to, and so on. To be included in a forward chain, one Execution Setup for the executable must be marked as Forward Chain Enabled and, if it should trigger additional downstream forward chaining, as Cascade Enabled. In addition, the executable's Work Area, Application Area, and Domain(s) must have forward chaining enabled.

Backchaining: Backchaining is a special type of processing in Oracle LSH where the system checks the data sources of a given executable to see if more current data exists in Oracle LSH or even in a source data system, and if so, executes all the Load Sets and Programs necessary to bring the most current data to the executable being submitted. You can define a Submission Type of Backchain for an Execution Setup so that a Program that operates on data generated by the current Program (for which you are defining the Execution Setup) can trigger the execution of the current Program as part of a backchain process.

Execution Setup Classifications: An Execution Setup has its own classifications. Its classifications determine where Oracle LSH displays the submission form in the Reports tab.

In addition, if a classification level value of any of the executable's Planned Outputs is set to Inherited, the system classifies the corresponding actual output using the classification value(s) of the Execution Setup for that level. The submission form and the actual output are then located in the same folder on the Reports tab.

An Execution Setup's classification values, if inherited, are inherited from its object (Program, Load Set, Data Mart, Report Set, or Workflow) instance. The object instance's inherited classification values are inherited from the Work Area.

See Classifying Objects and Outputs for instructions on classifying defined objects, including Execution Setups.

Execution Setup and Output Security: Execution Setups have their own user group assignments, inherited from their executable instance, which inherits its assignments from the Work Area. You can change the user group assignments if necessary; see Applying Security to Objects and Outputs .

Execution Setup user group assignments determine two things:

- The user groups' members can submit the Execution Setup to run the Program or other executable, if they have the necessary privileges within the user group.
- The user groups' members can view the output of the Program or other executable, if they have the necessary privileges within the user group.

  You can change an output's user group assignments after the output is generated. To do so, navigate to the output in the Reports tab or from the job ID on your Home Page and select Apply Security from the Actions drop-down.

If an output contains blinded or unblinded data, users needs special privileges to view it.

Execution Setup Versions: When you explicitly check out an Execution Setup, the system creates a new version of the Execution Setup. In addition, when you modify an Execution Setup, the system implicitly checks out the Execution Setup and gives it a new version number.

When an active Execution Setup is submitted, the system implicitly checks it in, if it is not already checked in; see Activating a Version of an Execution Setup.

> **Note:**
>
> Only the user who first created or modified the Execution Setup, and thereby checked it out, can undo the checkout. You can see that person's username in the **Checked Out By** field.

Execution Setup Upgrade: An Execution Setup must be synchronized with the definition of its executable object's Parameters. You can synchronize it at any time by selecting **Upgrade Execution Setup** from the **Actions** drop-down list in the Execution Setup screen. When you submit the Execution Setup and when you install its associated executable object, the system checks if an upgrade is required and performs it. If the Execution Setup is checked in, the system implicitly checks it out, upgrades it, and checks it in.

> **Note:**
>
> If the executable instance is modified to point to a different object definition, the system cannot automatically upgrade the Execution Setup. You must create a new Execution Setup.

Creating an Execution Setup

The system creates an Execution Setup automatically if you click **Submit** directly in the Properties screen of an executable object instance and no Execution Setup currently exists.

It is called Standard Execution Setup and is automatically set to be the default Execution Setup. In addition, you can create any number of Execution Setups manually as follows:

1. In the executable object instance for which you want to create an Execution Setup, select **Execution Setup** from the **Actions** drop-down list.

   The system opens the Execution Setup screen and displays any Execution Setups that have already been defined for this executable object.
2. Click **Create Execution Setup**. The Create Execution Setup screen opens.
3. Select one of the following options:

- Creating a New Execution Setup
  When you choose to create a new Execution Setup without an Execution Template, additional fields appear.
- Creating an Execution Setup from an Execution Template
  When you choose to use an Execution Template, a search field appears.

Creating a New Execution Setup

When you choose to create a new Execution Setup without an Execution Template, additional fields appear.

1. Enter values for the following fields:
   - **Name.** Give the Execution Setup a meaningful name, including an indication of the Program or other executable it is for.

> **Note:**
>
> The Execution Setup's name is very important because users submitting the executable from the Reports tab see only the Execution Setup name and description and executable object type (and its version number, validation status, and creation time). The Reports tab does not display information about the executable object to be run except for the object's type. Therefore the name/description combination must be informative enough to allow users to understand what they will get when they run the Execution Setup. If you plan to make this Execution Setup available as an Execution Template, remember that definers creating new Execution Setups based on this one will see only its name, its owning executable instance, and its validation and Runnable statuses.
>
> If there are or will be other Execution Setups for the same object, indicate what is unique about this one.

- **Description**. Describe the Execution Setup, especially if there are multiple Execution Setups for the same executable object.
- **Allow Use as Execution Template**. Set to **Yes** if you want to make this Execution Setup available as a template to other instances of the same executable definition. Set to **No** if you do not want the Execution Setup to be available as a template. You can change this setting only when the Execution Setup is checked in.

2. Select a subtype and change classifications if necessary. The output resulting from the submission of this Execution Setup may inherit its classifications from the Execution Setup; see Classifying Outputs.
3. Click **Apply**. The system generates a default Execution Setup using the current Parameter settings and returns you to the Execution Setup screen with the Execution Setup name displayed in the Execution Setup List.

   The current settings for Forward Chain Enabled and Cascade Enabled are displayed. You can change these settings by selecting the Execution Setup and clicking the appropriate button:

   - **Forward Chain Enabled**. If **Yes**, the Program or other executable can be triggered during a forward chain process or begin the forward chain process. Only one Execution Setup for the object can be set to Yes.
   - **Cascade Enabled**. If **Yes**, the Program or other executable can trigger the execution of executables that read from the tables it writes to as part of the forward chain process. Only one Execution Setup for the object can be set to Yes, and it must also have Forward Chaining Enabled set to Yes.

   > **Note:**
   >
   > Object types that do not write to tables, such as Data Marts, cannot have Cascade enabled.
   > Even if these attributes are set to Yes, if forward chaining is not enabled for the Work Area, this object cannot be executed as part of a forward chaining process.

4. Click the Execution Setup's name in the **Execution Setup** column.

   The system opens the Execution Setup Properties screen.

5. Modify as necessary. See Modifying an Execution Setup and Setting Parameters.
6. Set the Execution Setup version to Active; see Activating a Version of an Execution Setup.

Creating an Execution Setup from an Execution Template

When you choose to use an Execution Template, a search field appears.

1. Click the Search icon. The system displays all the Execution Setups available as Execution Templates that have been created for instances of the same execution object definition. For each Execution Template the system displays its Program instance name, validation status, and runnable status.
2. Click the icon in the Quick Select column for the Execution Template you want to use. The system returns you to the Create Execution Setup screen.
3. Click **Apply**. The system creates a new Execution Setup that is a copy of the Execution Template you selected and returns you to the Execution Setup screen with the Execution Setup name displayed in the lower portion of the screen.
4. Click the Execution Setup's name in the **Execution Setup** column.

   The system opens the Execution Setup Properties screen.

5. Modify as necessary. See Modifying an Execution Setup and Setting Parameters.
6. Set the Execution Setup version to Active; see Activating a Version of an Execution Setup.

Modifying an Execution Setup and Setting Parameters

After you have generated a default Execution Setup you can modify the system Parameters and the default values of the runtime Parameters. The Parameters in the Execution Setup are copies of those in the executable object definition. Modifying them has no effect on the object definition.

To appear in the Reports tab where it is available for general use, an Execution Setup must be checked in and set to Active (see Activating a Version of an Execution Setup). However, from the Execution Setup definition it is possible to submit the Execution Setup even if it is checked out.

See also:

- Removing an Execution Setup
- Activating a Version of an Execution Setup
- Submitting an Execution Setup

This section includes the following topics:

- Name and Description
  Click **Update** to modify the Execution Setup's Name and Description.

- Allow Use as Execution Template
  Set to **Yes** to make this Execution Setup available for use as a template to other instances of this Program. Set to **No** to make the Execution Setup unavailable as a template. This has no effect on any existing Execution Setups based on this one.
- System Parameters
  The same set of system Parameters are included in all Execution Setups.
- Runtime Parameters
  The system generates the runtime Parameters in an Execution Setup from the input and input/output Parameters, their default values and other settings, defined in the executable object instance.

Name and Description

Click **Update** to modify the Execution Setup's Name and Description.

> **Note:**
>
> The Execution Setup name is very important because it is what users see in the Reports tab. The name must be descriptive enough to allow users to understand what they will get when they run the Execution Setup.

Allow Use as Execution Template

Set to **Yes** to make this Execution Setup available for use as a template to other instances of this Program. Set to **No** to make the Execution Setup unavailable as a template. This has no effect on any existing Execution Setups based on this one.

System Parameters

The same set of system Parameters are included in all Execution Setups.

You can modify the Execution Setup to change the choices available to the person who uses the submission screen to submit a job. Each system Parameter is described below.

The standard system Parameters are described below.

To modify system Parameters, do the following:

1. Click the **System Parameters** subtab in the Execution Setup screen. The System Parameters screen opens.
2. Click the **Update** button in the System Parameters subtab.
3. You can modify the settings for any system Parameter as follows:
   - Select one of the following:
     - **Visible**. The Parameter is visible and its value is modifiable in the submission screen. This is the default value.
     - **Read Only**. The Parameter is visible but not modifiable in the submission screen.
     - **Hidden**. The Parameter is not visible or modifiable in the submission screen.

- **Allow**. Each system Parameter has a defined list of values. Select the checkbox for each value you want to make available to the user.
- **Default** (Optional). Set one of the allowed values as the default by clicking the radio button in the **Default** column next to the value. This value appears as the default value in the Submission screen.

> **Note:**
>
> If you are defining an Execution Setup for backchaining, for scheduled submission, or for triggered submission, you **must** define a default value for each Parameter, including the system Parameters.

The system Parameters are:

Submission Type: This system Parameter determines which submission methods this Execution Setup accepts:

- **Deferred**. A deferred submission runs once at a future scheduled time. Either you, working in the Execution Setup definition, or a user working in the Reports tab, can schedule the job to run one time in the future.
- **Immediate**. An immediate submission runs as soon as possible. Either you, working in the Execution Setup definition, or a user working in the Reports tab, can submit the job for immediate execution.
- **Scheduled**. A scheduled submission runs repeatedly on a regular schedule. Either you, working in the Execution Setup definition, or a user working in the Reports tab, can schedule the job to run on a regular basis in the future.
- **Backchain**. Backchain Execution Setups are called by another job—one that operates on data that has been loaded or generated directly or indirectly by this Program or Load Set—that has been submitted with the Data Currency Parameter set to Most Current Data Available. Users cannot submit a job for a backchain execution. They can, however, submit an immediate, deferred, or scheduled job that starts the backchain process.

  > **Note:**
  >
  > If you define a backchain Execution Setup, you must click **Submit** when you are finished. This creates a job of submission type Backchain that waits until triggered by a backchain process. The Execution Setup will not run during an actual backchain process unless you have already submitted it.

- **Triggered**. The Execution Setup accepts an XML message sent from an external system as the trigger for submission. For example, when Oracle Clinical batch validation has completed successfully, Oracle Clinical could send a message to trigger loading fresh data into Oracle Life Sciences Data Hub. Users cannot submit a job for a triggered execution.

> **Note:**
>
> When you define an Execution Setup to accept the Triggered submission type, allow Immediate and Deferred submission as well. The XML message must specify whether the job should be executed immediately upon receipt of the message or at a scheduled time in the future.

Notify on Completion: This setting determines when the person who submits a job using this Execution Setup receives a notification upon completion of the job:

- **Job Failure**. The submitter receives a Notification only if the job fails.
- **Never**. The submitter does not receive any Notification about the status of the job.
- **Job Success**. The submitter receives a Notification only if the job succeeds.
- **Job Warning**. The submitter receives a Notification only if the job ends with a status of Warning.

Data Currency: This system Parameter has the following allowed values:

- **Current Immediate Source**. This is the standard setting. The system processes the most current data contained in the source Table instances (or, in the case of a Load Set, the source data system).
- **Most Current Available (Triggers Backchain)**. This setting invokes a backchaining job that reexecutes all Program and Load Set instances that directly or indirectly feed data into the source Table instances of the current executable, if they meet the following criteria:
  - They have a submitted Execution Setup with a Submission Type of Backchain.
  - Each executable instance between them and the current executable also has an Execution Setup with a Submission Type of Backchain.
  - Their source data is more current than their target data.
- **Specify Snapshot**. This setting enables the user to specify a particular labeled or timestamped snapshot of source data at a previous point in time to run the Program against. If you select **Specify Snapshot**, you may also want to set the following:
  - **Default Snapshot Label**. From the drop-down list, select a snapshot label. This label will appear as the default value in the submission form. The drop-down list displays all snapshot labels shared by all source Table instances.
  - **Lock Default Snapshot Label**. If set to **Yes**, the person submitting the Execution Setup cannot change data currency; the Data Currency subtab is Read Only. If set to **No**, the submitter can change data currency in the Data Currency subtab.

Execution Priority: Set this value to determine the priority the system gives to the execution of this job, compared to other jobs of the same technology type. The choices are **High**, **Normal**, and **Low**.

Submission Mode: When the target Table instances require Reload or Transactional High Throughput processing, you must specify Full or Incremental Processing. The default value is Full.

> **Note:**
>
> The system does not display the Incremental options for jobs that do not have Reload or Transactional High Throughput Table instances as targets.

- **Incremental mode for Reload processing**. The system processes all records, compares the primary or unique key of each record to the current records in the table, and updates or inserts records as appropriate. Incremental mode never deletes records and therefore takes less time to complete.
- **Full mode for Reload processing**. The system does the same processing as in incremental mode and then performs the additional step of soft-deleting all records that were not reloaded. Full processing takes more time to complete but accurately reflects the state of the data in the source.
- **Incremental mode for Transactional High Throughput processing**. The system appends data to the Table after the last record. The system hard-deletes data if the Oracle Life Sciences Hub Program explicitly issues a Delete statement.
- **Full mode for Transactional High Throughput processing**. The system truncates the existing Table and loads fresh data into it. In this mode, you will lose all the Blinded data in the Table, even if you run the job with the Dummy setting for Blind Break.

> **Note:**
>
> The default data loading mode is Full. If you do not want to lose all your existing data, change the data loading mode to Incremental.

You may decide to use incremental mode on jobs that are run frequently, but use full mode at regular intervals on the same jobs when you need the most accurate data.

> **Note:**
>
> The submission screen contains additional system Parameters that allow the user to apply a snapshot label to target Table instances or to target and source Table instances at runtime. The user must also supply the text of the label at runtime. See "Setting Submission Details and Data Currency" in the *Oracle Life Sciences Data Hub User's Guide* for further information.

Blind Break: This Parameter is relevant only when one or more source Table instances either currently or formerly contained blinded data. Special privileges are required to run a job on real, sensitive, blinded data and to see the resulting output. The values available to the Definer in the Execution Setup depend on the current Blinding Status of the source Table instances:

- If none of the source Table instances has a Blinding Status, the only value available is **Not Applicable**. You may want to hide the Parameter (set it to **Hidden**) to avoid confusion.
- If one or more of the source Table instances has a Blinding Status of **Blinded**, the values available include **Real (Blind Break)** and **Dummy**. You can make the

Parameter Visible, allow both values, and make **Dummy** the default. Only users with Blind Break privileges on all blinded source Table instances will be able to select **Real (Blind Break)**.
If you are using DMW and all source tables with a Blinding Status of **Blinded** are blinded at the column, row, or cell level (not the table level), the **Dummy Blind Break** option allows processing the masking values in LSH. These are stored in the Dummy partition of the target table(s).

> **Tip:**
>
> Oracle recommends processing DMW data in DMW, not LSH. In particular, do not run DMW transformations and validation checks in LSH.

- If one or more of the source Table instances has a Blinding Status of **Unblinded**, and none of the source Table instances has a Blinding Status of **Blinded**, the values available include **Real (Unblinded)** and **Dummy**. You can make the Parameter Visible, allow both values, and make **Dummy** the default. Only users with Unblind or Blind Break privileges on all unblinded source Table instances will be able to select **Real (Unblinded)**.

> **Note:**
>
> If you set the default value to **Not Applicable** or **Dummy**, the System Parameters subtab displays the default value as **No**. If you set the default to **Real (Blind Break)** or **Real (Unblinded)** the System Parameters subtab displays it as **Yes**.

Force Execution: If set to **No**, before running a job based on this Execution Setup, the system compares the following for the current job and for the previous time the same instance was executed and does not reexecute the job if all of the following are true:

- The Program (or other executable) definition version is the same.
- The Parameter values are the same.
- The Source Job ID for the source Table instances is the same.

If all of the above are the same, then the same results should occur from running the job again. The system does not run the job and instead gives the user a message that it is a duplicate job. If any one of the above is different, the current job is not considered a duplicate of a past job and the system executes it.

If set to **Yes**, the system runs the job even if the data has not been refreshed, the Parameter settings are the same, and the Program version is the same.

> **Note:**
>
> In the case of a Report Set Execution Setup, the Report Set itself always executes, whether **Force Execution** is set to **Yes** or not. However, if **Force Execution** is set to **No**, the Programs in the Report Set do not run if the conditions outlined above are true. In addition, the **Force Execution** setting affects only Programs assigned to Report Set Entries that are selected for inclusion in the job.

Timeout Value: You can set a default timeout value—the length of time the system should continue to try to run the job after it has been submitted—by entering a number in the Default Value field and selecting a default unit of time (Minutes, Hours, Days, Weeks, Months). For example, if a report is scheduled to run once a day, a timeout period of 24 hours would prevent generating two reports for the same day. Alternatively, you can specify the allowed and default units of time without specifying a default value, and let the user set the value.

Apply Snapshot Label: The available values are:

- **None**. Users cannot apply a snapshot label to any Table instances at runtime.
- **Both**. Users can apply a snapshot label to both source and target Table instances at runtime.
- **Target**. Users can apply a snapshot label only to target Table instances at runtime.

Select one of these values to be the default. You can also specify a default label.

Runtime Parameters

The system generates the runtime Parameters in an Execution Setup from the input and input/output Parameters, their default values and other settings, defined in the executable object instance.

In the Runtime Parameters subtab the system displays each Parameter's default value (if any) and its Required flag setting. You can change the default setting that will appear in the submission screen by entering a value in the Parameter Value column.

If you want to view or change other settings, including Visible, Read Only, Required, and the validation method, click the Parameter's hyperlink. See Defining Variables and Parameters for further information.

> **Note:**
>
> If you add, remove, or modify Parameters in the executable object, when you install the object the system automatically upgrades its Execution Setups to reflect the change.

Predefined Runtime Parameters: Some object types have predefined runtime Parameters. In general, you should not change these Parameters in any way.

In particular, do not set values for the following Parameters in the Execution Setup:

- In Oracle Load Sets, do not enter a remote location. This is a security risk. The user submitting the Load Set for execution must have his or her own remote location/connection, which requires a username and password on the external Oracle system, or proper security access to a shared connection.
- In SAS and Text Load Sets, do not specify the file to be uploaded. The person submitting the Load Set for execution must enter a file name in the Server OS Filename parameter or the Data File Name parameter depending on whether Load From Server OS is Yes or No. The system uploads the file at the time that you specify it. If you specify it during Load Set or Execution Setup definition, the data in the file may be out of date by the time the user runs the Load Set.

Details about each predefined Parameter are included in the chapter about each object type: Setting Load Set Parameters and Setting Data Mart Parameter Values. Also see Setting Oracle BI Publisher Program Parameters.

Report Set and Workflow Execution Setups

For information on submitting a Report Set or Workflow for execution, see "Running a Job from Reports, My Home, or Applications" in the *Oracle Life Sciences Data Hub User's Guide*.

This section contains the following topics:

- About Report Set and Workflow Execution Setups
  Report Sets and Workflows have complex structures and contain other executable objects, so their Execution Setups are more complex than other objects'.
- Modifying a Report Set or Workflow Execution Setup
  This section contains information on the following sections of the screen:

About Report Set and Workflow Execution Setups

Report Sets and Workflows have complex structures and contain other executable objects, so their Execution Setups are more complex than other objects'.

Concurrent Editing: Many people can work on a Report Set or Workflow Execution Setup at the same time. However, only one person can work on a particular object or Parameter Set at a time.

Upgrading Execution Setup Structure and Parameters: The structure and Parameters of the Report Set or Workflow definition and the Execution Setup must remain synchronized. The system performs an automatic upgrade of the whole Execution Setup—synchronizing it with both the structure and the Parameters of the Report Set or Workflow—when the Report Set or Workflow is installed. In addition, when a Program instance in a Report Set or Workflow is installed, the system upgrades the Report Set or Workflow's Execution Setup to synchronize with the Program's Parameters.

You can trigger synchronization manually at any time by selecting **Upgrade Execution Setup** from the **Actions** drop-down list in the Execution Setup screen.

> **Note:**
>
> You must install a Program instance in a Report Set for the system to reflect changes to the Program's Parameters in the Report Set's Execution Setup.

Execution Setup Versioning: When a Report Set or Workflow is installed or submitted, the system implicitly checks in any of its Execution Setups that are not already checked in. When any user modifies any section of the Execution Setup, the system implicitly checks out the Execution Setup and gives it a new version number.

The system also checks in the Report Set or Workflow and all its Execution Setups when the Report Set or Workflow is copied.

Whenever the system performs an implicit checkin, if other people are working on any part of the Execution Setup, the system immediately checks it out again so that they can continue their work.

> **Note:**
>
> Only the user who first created or modified the Execution Setup, and thereby checked it out, can undo the checkout. You can see that person's username in the **Checked Out By** field for the Report Set-level Execution Setup.

Modifying a Report Set or Workflow Execution Setup

This section contains information on the following sections of the screen:

Execution Setup Properties: You can click **Update** and modify the following properties that belong to the Report Set Execution Setup as a whole:

- **Name** of the Execution Setup.
- **Description** of the Execution Setup.
- **Allow Use as Execution Template**. Set to **Yes** to make this Execution Setup available as a template for other Execution Setups in the Report Set definition.

Instances: This tab displays information a little differently for Report Set and Workflows:

- **Report Sets**. For Report Sets, you see the Report Set structure, with Report Set Entries displayed inside their parent Report Set Entries. You can **Expand All**, expand a particular node, or select a **Focus** icon to expand a particular node and hide the rest.
  Click any Report Set Entry's hyperlink to set the runtime Parameters for that Report Set Entry or its assigned Program instance.

  > **Note:**
  >
  > When you submit a Report Set for execution, you can specify which Report Set Entries to execute by using the checkboxes in the **Include** column.

- **Workflows**. For Workflows, you see all executable objects contained in the Workflow. Click any object's hyperlink to set its runtime Parameters.

The Instances subtab displays the following information:

- **Full Title**. For **Report Sets**, this column displays each Report Set Entry's concatenated full title that includes its Entry Number Prefix, Parent Number, Delimiter, Entry Number, Entry Number Suffix, and Title.
  For **Workflows**, this column displays the name of each executable object contained in the Workflow.
- **Ready**. If a checkmark in a green circle (the Ready icon) is displayed, all required Parameters associated with the object on that line have a value. For Parameters with a static list of values, the system checks if the value is valid. In Report Sets, the system checks all Report Set Entries that are children of the Report Set instance or Report Set Entry for which the icon is displayed; if any child Report Set Entry is not ready, the Not Ready icon is displayed for the parent as well.
  If an X in a red circle (the Not Ready icon) is displayed, at least one Parameter does not have a value, or has an invalid value.

  In the submission screen you can use the **Refresh** button to recalculate the status of all the **Ready?** flags at the same time, to verify that all interdependent values are valid.
- **Validation Status** of the object on that line.
- **Volume Name**. (Applies only to Report Sets) If the Report Set Entry begins a new volume of the Report Set, the system displays the volume name.
- **Program Name**. (Applies only to Report Sets) If a Program instance is assigned to the Report Set Entry, the system displays its name.
- **Program Checked Out By**. (Applies only to Report Sets) If the assigned Program instance is checked out, the system displays the username of the person who checked it out.
- **Planned Output Name**. (Applies only to Report Sets) If a Planned Output of the assigned Program instance is assigned to the Report Set Entry, the system displays its name.
- **Filename Reference**. (Applies only to Report Sets) If there is a Planned Output assigned, the system displays its filename.

Parameters: See Creating Parameters for Sharing Values within the Report Set.

Overlay Template Parameters: (Applies only to Report Sets) See Setting Overlay Template Parameter Values for information about each Parameter.

> **Note:**
>
> Overlay Template Parameters are displayed in the Execution Setup but not in the submission form.

Post-Processing Parameters: (Applies only to Report Sets) See Setting Post-Processing Parameter Values for information about each Parameter.

System Parameters: The system displays the current values for the Execution Setup's System Parameters; see System Parameters for information about each Parameter.

Removing an Execution Setup

You can remove an Execution Setup if you have delete privileges on Execution Setups of the relevant subtype and also Modify privileges on the parent object's subtype.

To remove an Execution Setup, do the following:

1. In the Applications tab, go to the object instance that contains the Execution Setup you want to remove.
2. Check out the object instance.
3. Select **Execution Setup** from the **Actions** drop-down list.
4. Click **Go**. The system opens the Execution Setup screen.
5. Select the Execution Setup you want to remove.
6. Click **Remove**. The system removes the Execution Setup.

Activating a Version of an Execution Setup

Only one version of an Execution Setup can appear in the Reports tab for submission at any one time.

Typically, it is the most recent version. However, you can select a different version if you want to. Whichever version you want to use, you must explicitly activate it or it does not appear in the Reports tab.

1. In the Applications tab, go to the object instance that contains the Execution Setup you want to activate.
2. Select **Execution Setup** from the **Actions** drop-down list.
3. Click **Go**. The system opens the Execution Setup screen.
4. Click the link to the Execution Setup. The system opens the Execution Setup's Properties screen.
5. From the **Actions** drop-down list, select **View Version History**.
6. Click **Go**. The system opens the Execution Setup Version History screen with the status of each version of the Execution Setup displayed in the Status column.
7. Select the version you want to be available for execution in the Reports tab and click **Set As Active**. The system changes the status of that version to **Runnable Active**.

> **Note:**
>
> You must also check in the Execution Setup on its Properties screen or it does not appear in the Reports tab.

Execution Setup Statuses: To see an Execution Setup's status, select **View Version History** from the **Actions** drop-down list. Execution Setups can have the following statuses:

- **Not Runnable**. An Execution Setup can be Not Runnable for the following reasons:
  - Its containing executable object instance has never been installed. To make the Execution Setup runnable, install the object instance.
  - Since the executable object instance was installed, one or more Parameters have been modified. To make the Execution Setup runnable, select **Upgrade Execution Setup** from the **Actions** drop-down list in the Execution Setup screen.
  - Changes have been made to the executable object instance (or its source definition) that are incompatible with the Execution Setup and that the system cannot automatically upgrade; for example, the executable instance now points to a different object definition. In this case, you cannot make the Execution Setup runnable. You must create a new Execution Setup.
- **Installable**. The containing executable object instance has been installed, then modified and checked in but not yet reinstalled. To make the Execution Setup runnable, install the instance. The system automatically upgrades the Execution Setup and changes its status to Runnable (unless the system cannot upgrade it, in which case its status changes to Not Runnable).
- **Runnable**. The Execution Setup is in synch with its containing executable object instance but has not been set to Active. To make the Execution Setup runnable, set it to Active.
- **Runnable Active**. The Execution Setup is in synch with its containing executable object instance and has been set to Active. It appears in the Reports tab. (Only one version of an Execution Setup can be set to Active.)

Submitting an Execution Setup

When you have created an Execution Setup, you can submit it to execute the Program, Load Set, Report Set, Workflow, or Data Mart, by clicking **Submit** in the Execution Setups screen or in the Reports tab.

You can then set any Parameters defined as Visible and not Read Only and schedule the job if the Execution Setup definition allows it. Instructions are included in "Generating Reports and Running Other Jobs" in the *Oracle Life Sciences Data Hub User's Guide*.

You can see the job's progress and details in the Jobs subtab of the object's Properties screen or the Job Execution section of your My Home screen. Click the Job ID to open the Job Details screen. Instructions for using the My Home screen are included in "Viewing Reports and Other Outputs" in the *Oracle Life Sciences Data Hub User's Guide*. You can also resubmit the Execution Setup from here.

If the job results in an output, you can see the output from the Job Details screen. If the output is classified, you can also see it in the Reports tab. Instructions for browsing and searching in the Reports tab are in "Viewing Reports and Other Outputs" in the *Oracle Life Sciences Data Hub User's Guide*.

Viewing Data

This section contains the following topics.

- Viewing Data within the Oracle Life Sciences Data Hub
  This section contains the following topics.
- Viewing Data with Visualizations
  After you have defined Business Areas, you can create instantaneous on-screen visualizations of the data specified in the Business Area.
- Viewing Data with Program-Generated Reports
  You can define a Program to display data in a report and run it at any time to see current data.
- Viewing Data Through an IDE
  After you have installed a Work Area and loaded data into it (or, in the case of Oracle data, created a view of data stored in an external system) you can use one of the integrated development environments (IDEs) to view data in Oracle LSH.

Viewing Data within the Oracle Life Sciences Data Hub

This section contains the following topics.

- About Data Browsing
  You can view a Table instance's data only if:
- Customizing Data Browsing
  If you do not want to see all the Table Columns, or if you have the required privileges and want to see blinded data, then click the **Customize** button on the Browse Data screen.

About Data Browsing

You can view a Table instance's data only if:

- The latest version of the Table instance is installed.
- You have Read Data privileges for the Table instance.
- For a blinded Table instance, you have blinding-related security privileges.

> **Note:**
>
> In a Work Area you can see whether or not a Table instance contains data. If the value in the **Has Data** column for the Table instance is **Yes**, it contains data; if it is **No**, it does not contain data.
> You can also see the latest and the installed Table instance version numbers in the Work Area's Properties screen.

Navigation: You can access the Browse Data feature in the following ways:

- **Work Area's Properties screen**. Click the icon in the **Browse Data** column. The icon is enabled only for Table instances whose latest version is installed.
- **Table instance's Properties screen**. Select **Browse Data** from the **Actions** drop-down list.

Data Currency: By default you see the current data.

You can select a timestamp from the **Data Currency** drop-down list to view data which is not current. The **Data Currency** drop-down list contains timestamps only if you have loaded data into the Table instance multiple times. You can see the Table instance's snapshot labels, if any, in parentheses next to the timestamp in the **Data Currency** drop-down list.

For blinded Table instances, the Data Currency drop-down list contains different sets of timestamps for dummy and real data. You can see the real data timestamps if you select **Real** from the **Blind Break** option in the Customize screen. See Customizing Data Browsing. You need the necessary privileges to use this feature.

Blinding: For Table instances that support blinding, the following rules apply:

- If the table contains blinded data, the system displays dummy data by default.
- If the table contains unblinded data and you have the required privileges, the system displays real data by default. If you do not have the privileges required to see unblinded data, the system displays dummy data.
- If the table does not support blinding, the system always displays real data.

Customizing Data Browsing

If you do not want to see all the Table Columns, or if you have the required privileges and want to see blinded data, then click the **Customize** button on the Browse Data screen.

You can customize the display through the following options:

- Columns
  When you click the **Customize** button for the first time, you can see the Table's Column names in an area titled **Included Columns**. Another area titled **Available Columns** contains no Column names.
- Where Clause
  You can use the **Filter Table instance where** feature to restrict the data you see.
- Order By Clause
  You can organize the data retrieved in any order using the **Order Table Instance by** feature.
- Blind Break
  The options you see in the **Blind Break** drop-down list depend on the state of the Table instance and on your privileges:

Columns

When you click the **Customize** button for the first time, you can see the Table's Column names in an area titled **Included Columns**. Another area titled **Available Columns** contains no Column names.

This is because by default, all Columns are selected for viewing. You can remove one or more Columns from view and also reorder them. After reordering, the Column on the top in the **Included Columns** area in the Customize Data Browse screen appears on the left in the Browse Data screen, and so on.
Do the following:

1. **To remove Columns from view**. Select the Column or Columns you want to remove from view and move them into the **Available Columns** area of the shuttle.

You can use Shift+Click or Ctrl+Click to select multiple Columns. You can double-click to move them or use the arrows.Enter the text of the first step here.

2. **To reorder columns.** Use the Up and Down arrows to reorder the Columns in the **Included Columns** area.
3. Click **Apply**.

You can now see the Columns you selected, in the order you selected.

Where Clause

You can use the **Filter Table instance where** feature to restrict the data you see.

To set up a Where filter:

1. Select either of the conditions to associate with a Table instance Column.
   a. Show results when ALL results are met
   b. Show results when ANY results are met
2. Select a Column from the drop-down list in the **Column** field. All available Columns are listed according to their position.
3. Select a filter condition from the drop-down list. The available conditions depend on the type of data in the particular Column.
4. Enter a value to filter by and select **Apply**. If the data type is VARCHAR 2, you can select the **Match Case** checkbox to filter values by case.

You can add different filter conditions to the same Column or view a combination of different Columns and conditions.
To include more Columns, select a Column from the drop-down list in the Add Another section and select **Add**. You can then apply filter conditions to the second Column. Similarly, use the **Remove** button to remove a Column and its filter conditions.

> **Note:**
>
> For example, for a Demography table with Columns Age, Sex and Race, you can set up the following Where filters:

- To retrieve data about patients who are over 50 years old, select: Column—Age, Condition—Greater than, Value—50
- To see data about female patients who are older than 50 years, select: Column—Age, Condition—Greater than, Value—50. Add another Column and select:

  Column—Sex, Condition—Is, Value—Female
- To see records for patients of all races except white, select: Column—Race, Condition—Is not, Value—White

Order By Clause

You can organize the data retrieved in any order using the **Order Table Instance by** feature.

You can specify an ascending or descending order for the Column's values. By default, the system displays records by sorting the values from the Column with an order number of 1 in ascending alphabetical or numerical order, depending on the Column's data type:

To order a Table instance:

1. Select a Column from the drop-down list in the **Column** field. Columns are listed according to their position in ascending order. Once selected, a Column does not appear in the list.
2. Select either Ascending or Descending from the drop-down condition list. Data is sorted in ascending order by default.
3. Select **Apply**.

You can add more Columns to order and view by using the choice list in the Add Another section, and selecting the **Add** button.
Use the **Remove** button to remove a column from the Table instance display.

For example, for a Demography table with Columns Age, Race, and Sex:

- To sort data by Age, select:
  Column—Age
- To sort data by Race, select:
  Column—Race
- To sort data in descending order of Age, and then ascending order of Sex, select:
  Column—Age, Condition—Descending. Add another column and select:

  Column —Sex

> **Tip:**
>
> You can find more information on SQL in these books:

You can find more information on SQL in these books:

Oracle® DatabaseSQL Language Reference at https://docs.oracle.com/cd/E11882_01/server.112/e41084/toc.htm

Oracle® Database PL/SQL Language Reference at https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm

Oracle® Database Reference at https://docs.oracle.com/cd/E11882_01/server.112/e40402/toc.htm

Blind Break

The options you see in the **Blind Break** drop-down list depend on the state of the Table instance and on your privileges:

- **Currently Blinded Table Instance**. If the table contains blinded data, you may see the following options:
  - **Dummy**. The system displays dummy data, not the real, sensitive data. This is the default behavior.
  - **Real (Blind Break)**. The system breaks the blind to show you the real, sensitive data. This option is available only if you have the necessary blinding-related privileges. The Table instance remains blinded.
- **Unblinded Table Instance**. If the table has been unblinded, you may see the following options:
  - **Real (Unblinded)**. The system displays the real, sensitive data that has been unblinded (made available to everyone with Read Unblind privileges). This option is available only if you have the necessary privileges. This is the default behavior for people with those privileges.
  - **Dummy**. The system displays dummy data, not the real, sensitive data. This is the default behavior for people who do not have Read Unblind privileges.
- **Table Instance without Blinding**. If the table does not support blinding (its **Blinding Flag** is set to **No**), the system always displays real data; the Table instance does not contain dummy data. The only option in the **Blind Break** drop-down list is **Not Applicable**.

Viewing Data with Visualizations

After you have defined Business Areas, you can create instantaneous on-screen visualizations of the data specified in the Business Area.

Viewing Data with Program-Generated Reports

You can define a Program to display data in a report and run it at any time to see current data.

See Defining Programs and Creating, Modifying, and Submitting Execution Setups.

Viewing Data Through an IDE

After you have installed a Work Area and loaded data into it (or, in the case of Oracle data, created a view of data stored in an external system) you can use one of the integrated development environments (IDEs) to view data in Oracle LSH.

To do this, you must set up a Program for this purpose:

1. Create a Program, including:
   - Source Table Descriptors mapped to the Table instances whose data you want to view.
   - A Source Code object. You do not need to upload any actual source code files.
2. Check in and install both the Program and the Table instances.
3. Check out the Program definition.
4. In the Program instance, launch the IDE.
5. Query the data as necessary.

For further information, see Defining Programs.

Viewing Jobs

You can see a record of each submission of an executable object instance from the Jobs subtab on its Properties screen.

The Jobs subtab shows jobs submitted by all users for all versions of the object instance.

You can set the **Show** filter to:

- **All**. Displays all submissions of the object instance.
- **Deferred**. Displays all submissions generated by Execution Setups with a single deferred execution.
- **Scheduled**. Displays all submissions generated by Execution Setups with an execution schedule.
- **Today's**. Displays all jobs with a Start TS of sysdate (the system ignores the time).

For each job, the system displays the following information. You can sort on any column by clicking the column header.

- **Job ID**. Click the hyperlink to see Job Execution Details. (Job execution details are explained in the My Home section of the *Oracle Life Sciences Data Hub User's Guide* chapter on tracking job execution.)
- **Submission Type**. The way the job was submitted for execution:
  - **Immediate**. The job was run as soon as it was submitted.
  - **Deferred**. The job was set up to run once at a later point in time from when it was submitted.
  - **Scheduled**. The job was one of series of regularly scheduled executions of the same executable object.
  - **Backchain**. The job was run as part of a backchain process ensuring that another job ran on the most current data available.
- **Job Status**. The highest job status the job has achieved. If the job is in progress, the status displayed is the current status. If the job has ended in failure or been aborted, the system displays the execution status the job had at the time it ended. For an explanation of job statuses, see the *Oracle Life Sciences Data Hub User's Guide* chapter on job execution.

> **Note:**
>
> To update the job status as the system processes the job, click **Refresh**.

- **User Name**. The name of the user who submitted the job.
- **Instance Version**. The version number of the object instance at the time of submission.
- **Execution Setup Name**. The name of the Execution Setup used for submission.
- **Submit TS**. The submission timestamp.

- **Start TS**. The timestamp of the time the job started.
- **End TS**. The timestamp of the time the job ended.

Viewing All Outputs of a Program or Report Set

You can view all existing outputs of executable objects several ways.

- Viewing All Outputs of an Instance in the Work Area Properties Screen
  You can see all outputs of a particular executable object instance from its Work Area.
- Viewing All Outputs from the My Home or Reports Tab
  You can also see outputs from the My Home and Reports tabs by clicking the Job ID.
- Viewing All Outputs of a Program or Report Set
  In addition to the preceding options, you can view all outputs of a Program or Report Set as follows.

Viewing All Outputs of an Instance in the Work Area Properties Screen

You can see all outputs of a particular executable object instance from its Work Area.

See View All Outputs for information.

Viewing All Outputs from the My Home or Reports Tab

You can also see outputs from the My Home and Reports tabs by clicking the Job ID.

For further information, see "Viewing Reports and Other Outputs" in the *Oracle Life Sciences Data Hub User's Guide*.

Viewing All Outputs of a Program or Report Set

In addition to the preceding options, you can view all outputs of a Program or Report Set as follows.

- Viewing All Program Outputs
  To view existing outputs for a Program, do the following.
- Viewing All Report Set or Report Set Entry Outputs
  To view existing outputs for a Report Set, do the following.
- Information Displayed
  For each Program, Report Set, or Report Set Entry output, the View Existing Outputs screen displays the following information.

Viewing All Program Outputs

To view existing outputs for a Program, do the following.

1. Navigate to a Program definition or instance.
2. In the Planned Output subtab, click the name of the Planned Output whose actually outputs you want to see. The selected Planned Output's Properties screen opens.

3. From the **Actions** drop-down list, select **View Existing Output**. The View Existing Outputs screen opens, listing all existing outputs.

> **Note:**
>
> The system displays existing outputs for all instances of the Program definition, even if you are in the Program instance's Properties screen. You can click on the Program Instance Name column heading to sort by Program instance.

4. Click the icon in the View column to open the actual output file.

Viewing All Report Set or Report Set Entry Outputs

To view existing outputs for a Report Set, do the following.

1. Navigate to a Report Set or Report Set instance Structure view screen.
2. Select the Report Set top level or the Report Set Entry whose outputs you want to see.
3. From the Select Object and: drop-down, select **View All Outputs** and click **Go**. The View Existing Outputs screen opens, listing all existing outputs.
4. Click the icon in the View column to open the actual output file.

Information Displayed

For each Program, Report Set, or Report Set Entry output, the View Existing Outputs screen displays the following information.

- File Name. Click the hyperlink to see the output's properties.
- Output Validation Status
- Creation TS (timestamp)
- Creation User (the username of the person who ran the Program instance that generated the output)
- Job ID. Click the hyperlink to see the job execution details.
- Program Instance Name
- Program Instance Version
- Path to Executable Instance (the location of the Program or Report Set instance)
- Title

Using the Actions Drop-Down List

You can use the **Actions** drop-down list to modify an object's classifications, security user group assignments, validation status, and more.

You can use the **Actions** drop-down list to modify an object's classifications, security user group assignments, validation status, and more.

Actions apply only to the object definition or the instance, not both. If you are in a Work Area, you are working on an object instance and the action applies only to the instance. To apply them to the definition, you must work in the definition directly in its Domain or Application Area.

Not all actions are available for both object definitions and instances. Also, not all actions change the object definition or instance; some actions only display specific information about the object definition or instance.

You do not need to click either **Update** or **Check Out**.

The following choices are available. Unless otherwise noted, they are available for both the definition (in its Domain or Application Area only) and the instance (in its Work Area) if you have the necessary privileges.

- **Assign Classification**. You can change the classification values assigned to the object. Classification values help users find the object using Advanced Search and in the case of instances, may affect the classification of the object's outputs. See Classifying Objects and Outputs for further information.
- **Apply Security**. Objects automatically inherit the user group assignments of their parent. You can remove those assignments and add others as necessary. See Applying Security to Objects and Outputs for further information.
- **Automatic Mapping by Name** (executable instances only). Map Table Descriptors to Table instances with the same name. See Automatic Mapping by Name for further information.
- **Execution Setups** (executable instances only). See Creating, Modifying, and Submitting Execution Setups for further information.
- **Reports**. You can generate reports that provide information about Table definitions and instances.
- **Support Validation Info**. You can change the instance's validation status and add outputs and documents as supporting information. See Validating Objects and Outputs for further information.
- **Table Descriptors from Existing Table Instances** (executable instances only). Create Table Descriptors from existing Table instances and map them at the same time. See Creating Table Descriptors from Table Instances and Simultaneously Mapping Them for further information.
- **Table Instances from Existing Table Descriptors** (executable instances only). Create Table instances from existing Table Descriptors and map them at the same time. See Creating Table Instances from Table Descriptors and Simultaneously Mapping Them for further information.
- **Upgrade All Instances** From the definition, you can upgrade any or all instances of the definition to the latest version of the definition. See Upgrading One or More Instances from the Definition for further information.
- **Upgrade Instance** (instances only). See Upgrading Object Instances to a New Definition Version for further information.
- **Version Label**. You can apply a label to the current version of the object definition or instance. See Version Labels for further information.
- **View All Outputs**. (Program definitions and instances only) Use this option to see all outputs generated by a particular Program definition or instance; see Viewing All Outputs of a Program or Report Set for further information.

- **View Version History** Use this option to see all previous versions of the definition. See Version History for further information.

Defining Planned Outputs

Different types of objects have different requirements for Planned Output definitions, which are placeholders for the actual outputs generated by the object during execution. See:

- Defining Planned Outputs
- Oracle Load Set Planned Outputs
- SAS Load Set Planned Outputs
- Text Load Set Planned Outputs
- OC Stable Interface Tables Load Set Planned Outputs
- OC DX SAS Views Load Set Planned Outputs
- OC Data Extract Views Load Set Planned Outputs
- OC Design and Definition Load Set Planned Outputs
- OC Global Metadata Load Set Planned Outputs
- OC Labs Load Set Planned Outputs
- OC Randomization Load Set Planned Outputs
- OC Study Data Load Set Planned Outputs
- Text Data Mart Planned Outputs
- SAS Data Mart Planned Outputs
- Oracle Export Data Mart Planned Outputs
- About Report Set Planned Outputs
- Workflow Planned Outputs

4

Defining Tables

This section contains information on the following topics.

- About Tables
  The tables that store data in the Oracle Life Sciences Data Hub are Oracle tables with additional metadata developed especially for Oracle Life Sciences Hub.
- Creating a Table
  When you create a Table in a Work Area, you are actually creating an instance of a Table definition.
- Setting and Modifying Table Attributes
  When you create a new Table definition and/or Table instance, the system populates its attribute values.
- Using the Table Properties Screen
  If you are working in a Work Area, you see the properties of both the Table instance and the Table definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.
- Defining Table Columns
  To create a new Table Column:
- Defining Table Constraints and Indexes
  This section contains the following topics.
- Modifying Tables
  If you have the necessary privileges, you can modify a Table either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area.

About Tables

The tables that store data in the Oracle Life Sciences Data Hub are Oracle tables with additional metadata developed especially for Oracle Life Sciences Hub.

The tables that store data in the Oracle Life Sciences Data Hub (Oracle LSH) are Oracle tables with additional metadata developed especially for Oracle Life Sciences Hub. Oracle Life Sciences Hub database tables are designed to hold data that originated in either a SAS or an Oracle system, or both. They are designed so that computer programs based on either Oracle or SAS technology can read from or write to any Oracle Life Sciences Hub table. Oracle Life Sciences Hub tables have the following special characteristics:

- **Both Oracle and SAS Metadata Attributes**. Oracle Life Sciences Hub database tables have Oracle and SAS names and a SAS label at both the table/data set and column/variable levels.
- **Blinding Attributes**. Oracle Life Sciences Hub tables have special attributes for use in maintaining securely blinded clinical data. See Managing Blinded Data for further information.

- **Data Processing Attribute**. Oracle Life Sciences Hub tables have a special attribute whose value determines how the system processes data written to each table. Most methods of data processing result in an audit trail of all data changes within the table. See Data Processing Types.
- Table Definitions and Instances
  You must create both a Table definition and a Table instance, and install the instance in the database.
- Reading from and Writing to Tables
  There are several types of objects that can read data from and/or write data to an Oracle Life Sciences Data Hub Table instance, as follows.
- Creating Tables
  As you develop applications in Oracle Life Sciences Data Hub, you can create Oracle Life Sciences Hub Tables in the following ways.
- Columns
  Table Columns are instances of Variables, as are Parameters.
- Reports on Table Definitions and Instances
  From the **Actions** drop-down list, you can generate reports that provide information on a Table definition or instance.

Table Definitions and Instances

You must create both a Table definition and a Table instance, and install the instance in the database.

See Object Definitions and Instances, and their Containers for further information.

The Table definition is a metadata representation of a Table that includes most of the detailed specifications of the Table, including the number of Columns, the name, data type and length of each Column, and primary key, unique key, and check constraints.

The Table instance contains a pointer to the Table definition and a few details such as a mapping to at least one Table Descriptor contained in a Program or other executable instance. To actually store data in a table, you must install the Table instance in the database. The installation process instantiates the metadata from both the Table instance and the Table definition to which it points in the database to create a database table. Some attributes belong to both the definition and the instance. In this case, you can overwrite the value in the instance, and the installation process creates a database table with the value specified for the instance. As with other object definitions in Oracle Life Sciences Data Hub, you can create multiple instances of the same Table definition.

Reading from and Writing to Tables

There are several types of objects that can read data from and/or write data to an Oracle Life Sciences Data Hub Table instance, as follows.

Each type contains a Table Descriptor for each Table instance they either read from or write to. The Table Descriptor and Table instance must be mapped. For further information, see Defining and Mapping Table Descriptors.

- **Programs**. Programs can read from and/or write to installed Oracle Life Sciences Hub Table instances. Report Sets and Workflows contain Programs, and therefore

they can indirectly read from and/or write to Table instances. Only one Program instance can write to a particular Table instance.

- **Load Sets**. A Load Set loads data from an external system into one or more installed Oracle Life Sciences Hub Table instances (see Defining Load Sets).
- **Data Marts**. A Data Mart reads data from an installed Oracle Life Sciences Hub Table instance and copies the data into a file (see Defining Data Marts).
- **Business Areas**. A Business Area contains Table Descriptors that allow a data visualization tool to read data from Oracle Life Sciences Hub Table instances.

Creating Tables

As you develop applications in Oracle Life Sciences Data Hub, you can create Oracle Life Sciences Hub Tables in the following ways.

Working in a Work Area, you can:

- Create a Table instance from an existing Table definition
- Create a new Table instance and Table definition manually at the same time
- Create a Table instance and Table definition from an existing SAS data set located in an external system

Working in a Program, Load Set, Data Mart, or Business Area you can:

- Create a Table Descriptor from an existing Table definition
- Create a Table Descriptor and Table definition manually at the same time
- Create a Table Descriptor from an existing SAS data set located in an external system
- Create one or more Table instances from one or more Table Descriptors (using the **Actions** drop-down list)
- Create one or more Table Descriptors from one or more Table instances (using the **Actions** drop-down list)

In addition, working in a Load Set you can create a Table Descriptor from a table, data set, or view in an external system.

You can also create a Table definition directly in a Domain or Application Area if you have the necessary privileges. However, you cannot use the definition for storing data until you create an instance of it in a Work Area and install it in the database.

Columns

Table Columns are instances of Variables, as are Parameters.

See Defining Variables and Parameters for further information.

Reports on Table Definitions and Instances

From the **Actions** drop-down list, you can generate reports that provide information on a Table definition or instance.

See System Reportsfor further information.

Creating a Table

When you create a Table in a Work Area, you are actually creating an instance of a Table definition.

To create a new Table instance:

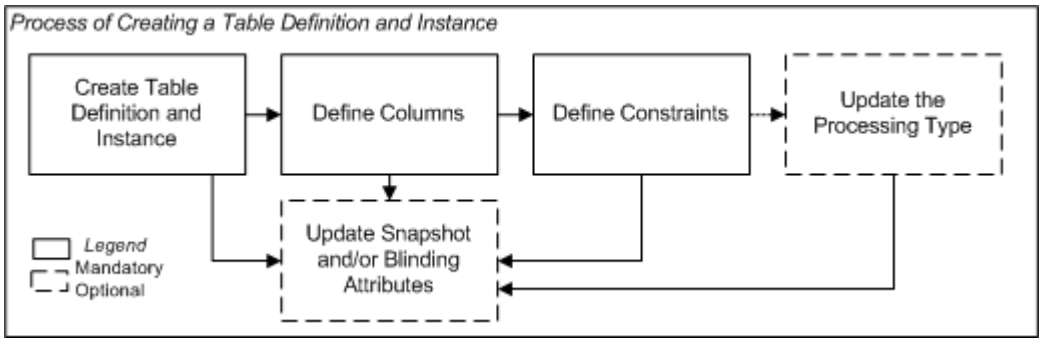
1. In a Work Area, select **Table** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Table screen.
3. Choose one of the following options:
   - **Create new table definition and instance**. Choose this option if no Table definition exists that can meet your needs, either as it is or with some modification.
   - **Create an instance of an existing Table definition**. Choose this option if an Oracle Life Sciences Hub Table definition already exists that meets your needs.
     If you can adapt an existing Table definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.
   - **Create new table definition and instance from SAS data set**. Choose this option if you want to create an Oracle Life Sciences Hub Table with the same metadata structure as an existing SAS data set.
4. Depending on your choice, follow one of the following sets of instructions:
   - Creating a New Table Definition and Instance
   - Creating an Oracle Life Sciences Data Hub Table From a SAS Data Set
   - Creating a New Instance of an Existing Table Definition
5. Modify the default attribute settings as necessary; see Setting and Modifying Table Attributes.
6. Define Columns as necessary; see Defining Table Columns .
7. Define Constraints and Indexes as necessary; see Defining Table Constraints and Indexes .

- Creating a New Table Definition and Instance
- Creating an Oracle Life Sciences Data Hub Table From a SAS Data Set
  You can upload SAS data set metadata into Oracle Life Sciences Hub to create a Table with the same structure as a data set in an integrated external system. The system creates Table Columns with the same data type and length as the variables in the data set.
- Creating a New Instance of an Existing Table Definition
  If you use an existing Table as a definition source, its Columns and other properties are already defined.

Creating a New Table Definition and Instance

Figure 4-1 Process of Creating a Table Definition and Instance



When you select **Create new table definition and instance** in the Create Table screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Description**. See Creating and Using Object Descriptions.
   - **Oracle Name** (up to 30 characters, uppercase, no spaces). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces, truncated to 30 characters if necessary.
   - **SAS Name** (up to 32 characters, uppercase, no spaces). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces, truncated to 32 characters if necessary.
   - Enter a **SAS Label** (optional, up to 256 characters). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field.
2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Table.

   The system opens the Properties screen for the new Table instance.
4. Define the details. See:
   - Setting and Modifying Table Attributes
   - Defining Table Columns
   - Defining Table Constraints and Indexes

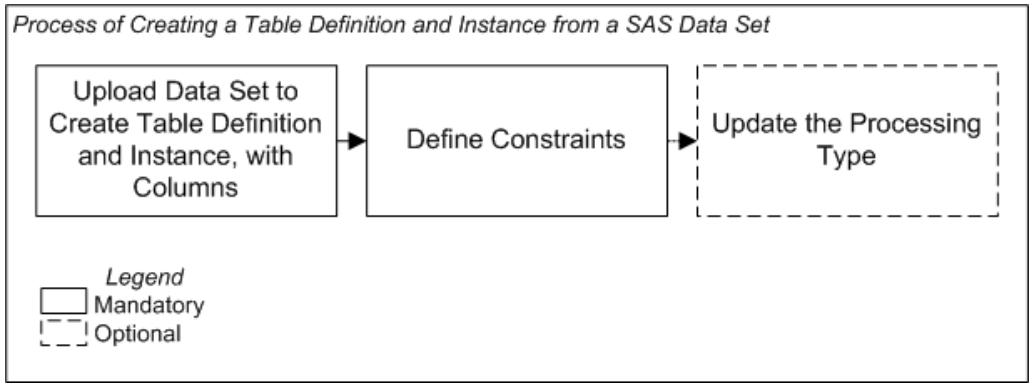
Creating an Oracle Life Sciences Data Hub Table From a SAS Data Set

You can upload SAS data set metadata into Oracle Life Sciences Hub to create a Table with the same structure as a data set in an integrated external system. The system creates Table Columns with the same data type and length as the variables in the data set.

> **Note:**
>
> Oracle Life Sciences Hub gives SAS variables of SAS format BEST8 a length of 8 and Precision set to null.

The system searches for Variables in the same Application Area with the same name, data type, and length as each of the variables in the data set. If a matching Variable exists, the system bases a Column of the Table definition on it. If a Table definition with the same name already exists in the Application Area, the system appends _1 to it, or _x if the Table name already has a number appended, where x is the next larger integer.

Figure 4-2 Process for creating a SAS Data Set



When you select **Create new table definition and instance from SAS data set** in the Create Program screen, additional fields appear.

1. Click **Browse**. A standard Choose file pop-up window opens.
2. Select the .sas file on a local or shared drive and click **Open**.
3. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
4. Click **Apply** to save your work and continue defining the Table.

   The system opens the Properties screen for the new Table instance.

5. Define the Table details in the lower part of the screen. See:
   - Defining Table Columns
   - Defining Table Constraints and Indexes

> **Note:**
>
> You can also create an Oracle Life Sciences Hub Table from a SAS data set when you define Load Set Table Descriptors. You can then use the Actions drop-down list item **Table Instance from Existing Table Descriptor** to create an identical target Table instance and map the two.

Creating a New Instance of an Existing Table Definition

If you use an existing Table as a definition source, its Columns and other properties are already defined.

See Creating an Instance of an Existing Definition.

Setting and Modifying Table Attributes

When you create a new Table definition and/or Table instance, the system populates its attribute values.

To change these values, as well as the name and other attributes described in Creating a New Table Definition and Instance, click **Update**.

- Process Type
  The processing type determines how Oracle Life Sciences Data Hub writes data to the Table instance.
- Allow Snapshot
  Snapshots are possible only on audited Tables (Reload, Transactional with Audit, or Staging with Audit).
- Blinding Flag
  This setting has effect only for Table instances.
- Blinding Status
- Definition Source
  This field applies to the instance only.
- Tablespace Name
  If your company has created its own tablespaces, select the appropriate tablespace for the actual database table created when you install this Table instance.

Process Type

The processing type determines how Oracle Life Sciences Data Hub writes data to the Table instance.

See Data Processing Types for information.

For Table instances that are the target of an Oracle-technology Load Set, you can instead select Create Table as a View from the Process Type drop-down list. No processing type is required because the system does not write data to the Table instance. Instead, you use the Table instance as a pass-through view to see data in the source system. The option appears after you map the Table instance to an Oracle-technology Load Set.

> **Note:**
>
> The system prevents you from selecting a processing type that requires a primary or unique key (Reload or Transactional with Audit) until you have done the following:
>
> - Defined Columns for the Table
> - Defined a primary or unique key for the Table
> - In the Audit Key Constraint field, which appears when you select a processing type that requires a primary or unique key, selected the primary or unique key you want the system to use for auditing. See Data Auditing.

Allow Snapshot

Snapshots are possible only on audited Tables (Reload, Transactional with Audit, or Staging with Audit).

See Data Snapshots.

- If set to **No**, creating and labeling snapshots of this Table instance is not allowed.
- If set to **Yes**, users can create and label snapshots of this Table instance.

Blinding Flag

This setting has effect only for Table instances.

The setting in the Table definition serves only as a default setting for instances of that Table definition. See Managing Blinded Data for further information.

- If set to **No**, the Table instance is intended to never contain blinded data.
- If set to **Yes**, the Table instance may contain blinded data. The system maintains two sets of rows: one set for the real data and another set for dummy data, effectively partitioning the table in the database.

> **Note:**
>
> If you change the setting of the Blinding flag for a Table instance that is already installed, you cannot use an Upgrade-mode Work Area installation to reinstall the new version of the Table instance. You must do a Full-mode Work Area install to apply a new blinding status to an installed Table instance, which deletes the Table instance's data; see About Work Area Installation.

Blinding Status

If the Blinding Flag is set to **Yes**, you can set the Blinding Status attribute to one of two values:

- **Blinded**. The real, sensitive data cannot be viewed or operated on except by a user with Blind Break privileges. Any user with normal security access to the Table instance can view and operate on the dummy data.
- **Unblinded**. A person with special privileges can change the Blinding Status from Blinded to Unblinded; for example, at the end of a trial. After the status is set to Unblinded, users require special, but less restrictive, privileges to view and operate on the unblinded data. See the chapter on security in the *Oracle Life Sciences Data Hub Implementation Guide* for more information on blinding-related privileges. See also Managing Blinded Data.

If the Blinding Flag is set to **No**, you can set the Blinding Status attribute to one of two values:

- **Not Applicable**. This is the default value. It is intended for use with Table instances that never will contain sensitive information that would require blinding.
- **Authorized**. Use this status in the rare case that the Table instance is the target of a Program that reads from a blinded Table instance, and the Program is written in such a way that this target Table instance will never contain sensitive data that should be blinded. When the Blinding Status is set to **Authorized**, the system allows users who have Blind Break privileges on all blinded source Table instances to run the Program that writes to this Table instance. These users see a warning and must confirm that they want to run the Program.
  You may need to use this feature, for example, when creating dummy data; see Loading Real and Dummy Data.

> **Note:**
>
> While you are developing a Program that reads from a blinded Table instance and writes to a nonblinded Table instance, set the target Table instance's Blinding flag to **Yes** until you are sure that neither the Program nor its log file exposes any sensitive information.

Definition Source

This field applies to the instance only.

It specifies the Table definition to which this Table instance points. See Definition Source.

Tablespace Name

If your company has created its own tablespaces, select the appropriate tablespace for the actual database table created when you install this Table instance.

If you do not enter a value, the system creates the table in the default tablespace for the database.

1. Click the Search icon. The Search and Select window opens.
2. Enter the name of the tablespace you are looking for and click **Go**. You can use the % wildcard, but not as the first character. The system displays the tablespaces that meet the search criteria.
3. Click the Quick Select icon. The system returns you to the Table instance screen with the selected tablespace displayed.

> **Note:**
>
> The list of tablespace name values is stored in the lookup CDR_TABLESPACE_NAMES. For information on adding values, see " Adding, Modifying, or Discontinuing a Lookup Value" in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

Using the Table Properties Screen

If you are working in a Work Area, you see the properties of both the Table instance and the Table definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See Modifying Tables for information on modifying Tables.

See also Figure 4-1.

This section contains the following topics:

- Instance Properties
  You can see the following instance properties:
- Definition Properties
- Buttons
  From a Table instance in a Work Area, you can use the following buttons:

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Oracle Name:** This is the Table's name to be used in PL/SQL code.
- **SAS Name:** This is the Table's name to be used in SAS code.
- **SAS Label:** This is an optional field. You may see a SAS label for the Table here. See Setting and Modifying Table Attributes for a description of the following:
  - Process Type
  - Allow Snapshot
  - Blinding Flag
  - Blinding Status

  - Definition Source
  - Tablespace Name
- **Validation Status:** This field displays the current validation status of the Table instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Table: Installable or Non Installable. See Installation Requirements for Each Object Type.
- Version: This field displays the current version number of the Table instance.
- Version Label: This field displays the version label, if any, for the current Table instance version.
  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.

Definition Properties

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to modify Columns and Constraints/Indexes. See Understanding Object Versions and Checkin/Checkout for further information.
- **Latest Version:** If set to **Yes**, this Table instance is pointing to the latest version of the Table definition. If set to **No**, this Table instance is pointing to an older version of the Table definition.
- **Checked Out By:** This field displays the username of the person who has the Table definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Validation Status:** This field displays the current validation status of the Table definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Table in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Table: Installable or Non Installable. See Installation Requirements for Each Object Type.

Buttons

From a Table instance in a Work Area, you can use the following buttons:

- **Update:** Click **Update** to modify the Table instance properties. See Modifying Table Instance Properties.
- **Check In/Out and Uncheck:** Click these buttons to check out, check in, or uncheck the Table definition. Different buttons are displayed in the Table Definition Properties section depending on the Checked Out Status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of the person

who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.

Defining Table Columns

To create a new Table Column:

1. In the Columns subtab of a Table, click **Add**. The system displays the **Create Column** screen.
2. Choose one of the following options:
   - Create a New Column and Variable. Choose this option if no Variable definition exists that meets your needs. See Create a New Column and Variable.
   - Create a Column from an Existing Variable. Choose this option if a Variable already exists that meets your needs. Using this option whenever possible increases consistency and reusability in your Oracle Life Sciences Data Hubapplications.
     See Finding an Appropriate Definition and Reusing Existing Definitions for further information. For instructions, see Creating an Instance of an Existing Definition.

- Create a New Column and Variable
  To create a new Column definition and instance at the same time, enter the following information:
- Create a Column from an Existing Variable
  You can create individual Columns as instances of existing Columns, as you can for other objects.

Create a New Column and Variable

To create a new Column definition and instance at the same time, enter the following information:

1. Enter values in the following fields:
   - **Name**. See Naming Objects.

     > **Note:**
     >
     > Do not use special characters in column names, except underscore (_). If you do use other special characters, Oracle Life Sciences Data Hub automatically converts them to underscores in the SAS Name, whose default value is derived from the Name. For example, if you upload a column with the name COL$UMN, Oracle Life Sciences Hub automatically converts the name to COL_UMN in the SAS Name.

   - **Description**. See Creating and Using Object Descriptions.
   - **Data Type**. Select one of the following from the drop-down list.
     - **VARCHAR2**. Specifies a variable-length character string. For each row, the system stores each value in the Column as a variable-length field

unless a value exceeds the Column's maximum length, in which case the system returns an error.

  - **NUMBER**. Stores zero, positive, and negative fixed and floating-point numbers. A Number Column can contain a number with or without a decimal marker and/or a sign (-). All standard rules for the Oracle Number data type apply.
  - **DATE**. For each Date value, Oracle stores the following information: century, year, month, date, hour, minute, and second. Although date and time information can be represented in both character and number datatypes, the Date datatype has special associated properties.

- **Length**. The maximum number of bytes or characters of data that the Column can hold. The requirements vary according to the data type:
  - **VARCHAR2**. A value for length is required and must be between 1 and 4000 characters.
  - **DATE**. The system disregards the length value, if any.
  - **NUMBER**. A value for length is optional.You can leave the length and precision null, and Oracle LSH treats the number column as having the maximum possible length.
- **Precision**. (This field appears only if you select a data type of NUMBER.) The total number of digits allowed. For example, if Precision is set to 2 and a data value is 34.333 is entered in this Column, the system stores the data value as 34.33. Oracle guarantees the portability of numbers with precision ranging from 1 to 38.
- **Oracle Name**. Name to use for the Column in PL/SQL source code. The value defaults from the Name value, converted to uppercase and with underscores substituted for spaces. You can change the default value.
- **SAS Name**. Name to use for the Column in SAS source code. Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces.
- **SAS Label** (Optional) Enter up to 200 characters.
- **SAS Format** Required:
  - For columns of VARCHAR2 data type, by default, the system enters a dollar sign ($) followed by the value you entered in the Length field.
  - For columns of NUMBER data type, by default, the system enters a period/full stop (.).
- **Default Value** (optional). You can enter a value to serve as the default for this Column.
- **Nullable**. If set to **Yes**, null values are allowed in this column. If set to **No**, each row must have a value in this column.

2. In the **Classification** section, select the following for the Variable:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Table.

   The system opens the Properties screen for the new Table instance.

Create a Column from an Existing Variable

You can create individual Columns as instances of existing Columns, as you can for other objects.

In the case of Columns, a Column definition is really a Variable.

For instructions, see Creating an Instance of an Existing Definition.

Defining Table Constraints and Indexes

This section contains the following topics.

- About Constraints
  You can define Constraints to enforce limitations on data in each row of a table.
- Check Constraint
  The check Constraint allows you to specify allowable values for a particular Column.
- Non-Unique Index
  You can define a non-unique index on any Column or set of Columns.
- Primary Key
  A Primary Key is a Column or set of Columns whose value(s) identify a row in a table as unique.
- Unique Key
  A Unique Key is similar to a Primary Key in that it can include one or more Columns whose value(s) identify a row as unique.

About Constraints

You can define Constraints to enforce limitations on data in each row of a table.

You can define Constraints to enforce limitations on data in each row of a table. Oracle Life Sciences Data Hub automatically generates an index based on the Primary Key Constraint. Oracle Life Sciences Hub Constraints and indexes appear as Oracle table constraints and indexes for Oracle technology Programs and as SAS data set constraints and indexes for SAS technology Programs.

You must define the Table's Columns before you can define Constraints.

Constraints are different from other Oracle Life Sciences Hub defined objects in that they belong to the Table instance as well as to the definition. If you define or modify a Constraint in a Table instance in a Work Area, the resulting Constraint is part of the Table instance but not the Table definition on which it is based. The only way to make a Constraint a part of the Table definition is to go to the definition directly in the Application Area or Domain and add it there. After you have added a Constraint to a Table definition, all Table instances created from it also have that Constraint defined. You can delete the Constraint from the Table instance if you want to.

You must define a Primary or Unique Key for a Table in order to perform most types of data processing (see Data Processing Types).

> **Note:**
>
> Oracle Life Sciences Hub supports the standard Oracle Not Null constraint, but you define it as a Column attribute, not through the Table Constraints user interface.

To create a Constraint:

1. In the Constraints/Indexes subtab of a Table, click **Add**.

   The system displays the **Create Constraint** screen.
2. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Description**. See Creating and Using Object Descriptions.
   - **Type**. Choose the type of Constraint. The system refreshes the display of the lower portion of the screen depending on which type you choose. The definition procedure varies by type: see Check Constraint, Primary Key, Unique Key.
3. Click **Apply**.

Check Constraint

The check Constraint allows you to specify allowable values for a particular Column.

For example, you can require that a particular Column contain either a Yes or No value in every row.

If any row contains a different value for the Column, the system generates an error to the Program writing to the Table instance. If the Program does not handle the error, the job fails.

> **Note:**
>
> Oracle does not verify that check Constraints are not mutually exclusive. Therefore, if you create multiple check Constraints, design them carefully so their purposes do not conflict. Do not assume any particular order of evaluation of the conditions.

To define a Check Constraint:

1. Select **Check Constraints** from the **Type** drop-down list. The system refreshes the lower portion of the screen.
2. From the **Column Name** drop-down list, select the Column for which you want to define a list of allowable values.
3. In the **Column Values** section, enter one allowable **Check Value** value in the first row.
4. To add an additional row, click **Add Another Row**.

5. Click **Apply**.

Non-Unique Index

You can define a non-unique index on any Column or set of Columns.

Unlike a Primary or Unique Key, a non-unique index does not validate each row for a unique value but allows different rows to have the same value in the Column or set of Columns that are part of the index.

Like other indexes, a non-unique index keeps rows sorted on the specified Column or Columns so that the system can use faster search algorithms on the table, speeding up queries on the table.

To define a non-unique index, do the following:

1. Select **Non-Unique Index** from the **Type** drop-down list. The system refreshes the lower portion of the screen and lists all the **Columns** you have defined in the current Table instance in the **Table Columns** side of the shuttle.
2. Select the **Column** or **Columns** you want to be part of the unique key and move them into the **Non-Unique Index** side of the shuttle. You can use Shift+Click or Ctrl+Click to select the **Columns**. You can double-click to move them or use the arrows.
3. If necessary, use the **Up** and **Down** arrows to reorder the Columns in the key. You should have the most general Column at the top and the most granular at the bottom. For example, if your non-unique key is Patient, Visit, Test, you should list them in that order.
4. Click **Apply**.

Primary Key

A Primary Key is a Column or set of Columns whose value(s) identify a row in a table as unique.

A single-Column Primary Key is commonly a unique ID. A multi-Column, or **composite**, Primary Key might include, for example: Patient, Visit, Test, if each test was conducted only once at a particular visit. None of the Columns that is part of a Primary Key can have a null value in any row.

The system automatically creates an index based on the primary key, which it uses to enforce a unique constraint. The index also speeds up queries on the table.

When you define a Primary Key, the system enforces constraints on the data:

- The Column value(s) must serve as a unique identifier for each row.
- No value in a key Column can be null. Columns that are part of a Primary Key must have the Nullable attribute set to **No** (the check box must be cleared).

To define a Primary Key:

1. Select **Primary Key** from the **Type** drop-down list. The system refreshes the lower portion of the screen and lists all the Columns you have defined in the current Table instance in the Table Columns side of the shuttle.
2. Select the **Column** or **Columns** you want to be part of the primary key and move them into the **Primary Key** side of the shuttle. You can use Shift+Click or Ctrl

+Click to select the **Columns**. You can double-click to move them or use the arrows.

3. If necessary, use the **Up** and **Down** arrows to reorder the Columns in the key. You should have the most general Column at the top and the most granular at the bottom. For example, if your Primary Key is Patient, Visit, Test, you should list them in that order so that you can use the automatically generated index.
4. Click **Apply**.

Unique Key

A Unique Key is similar to a Primary Key in that it can include one or more Columns whose value(s) identify a row as unique.

The difference is that the system allows null values in the Columns that are part of a Unique Key.

Any number of rows can include null (empty) values. A null in a Column (or even all Columns where allowed in a composite Unique Key) satisfies the Unique Key Constraint. However, you cannot have identical non-null values in the Columns of a partially null composite Unique Key Constraint.

If, as part of the definition of a Column that is part of a Unique Key, you disallow null values for that Column, the system does not allow null values there.

The Unique Key also serves as an index.

To define a Unique Key:

1. Select **Unique Key** from the **Type** drop-down list. The system refreshes the lower portion of the screen and lists all the **Columns** you have defined in the current Table instance in the **Table Columns** side of the shuttle.
2. Select the **Column** or **Columns** you want to be part of the unique key and move them into the **Unique Key** side of the shuttle. You can use Shift+Click or Ctrl+Click to select the **Columns**. You can double-click to move them or use the arrows.
3. If necessary, use the **Up** and **Down** arrows to reorder the Columns in the key. You should have the most general Column at the top and the most granular at the bottom. For example, if your Unique Key is Patient, Visit, Test, you should list them in that order so that you can use the key as an index.
4. Click **Apply**.

Modifying Tables

If you have the necessary privileges, you can modify a Table either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area.

In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Table Instance Properties
  On the Table instance's Properties screen, click **Update** to enter changes. Oracle Life Sciences Data Hub creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.
- Modifying Table Definition Properties
  You can go to a Table definition's Properties screen in one of the following ways:

Modifying Table Instance Properties

On the Table instance's Properties screen, click **Update** to enter changes. Oracle Life Sciences Data Hub creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must reinstall the Table for the changes to take effect.

You can modify the following:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field applies to the instance only. It specifies the Table definition to which this Table instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Columns and Constraints/ Indexes.
  - The Status of the Table changes to Non Installable.

  If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.

  See Setting and Modifying Table Attributes for a description of the following:
  - Process Type
  - Allow Snapshot
  - Blinding Flag
  - Blinding Status

  - Definition Source
  - Tablespace Name

Modifying Table Definition Properties

You can go to a Table definition's Properties screen in one of the following ways:

- From the **Table's Properties screen**: Click the hyperlink of the Table definition that appears in the Definition field. See Definition Source.
- From the **Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Table definition screen, click **Update** to enter changes. Oracle Life Sciences Data Hub creates a new version of the definition. You can change the following properties:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information. You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

  See Setting and Modifying Table Attributes for a description of the following:

  - Process Type
  - Allow Snapshot
  - Blinding Flag
  - Blinding Status
  - Definition Source
  - Tablespace Name
- Modifying Columns
  To modify a Column in any way, you must check out the Table definition; Columns belong to the Table definition.
- Modifying Constraints and Indexes
  When you create a Table instance from a Table definition, the system copies the definition's constraints and indexes to the instance instead of referencing them there.

Modifying Columns

To modify a Column in any way, you must check out the Table definition; Columns belong to the Table definition.

You can delete and add Columns, and change the definition source Variable for a Column (the source Variable determines the Column's data type, length, and default value, if any). You cannot modify the data type, length, or default value except by substituting a different Variable as the definition source.

Modifying Constraints and Indexes

When you create a Table instance from a Table definition, the system copies the definition's constraints and indexes to the instance instead of referencing them there.

Therefore, any changes you make to the Table instance in the Work Area affect only the instance.

Normal Oracle database rules apply to making changes in table constraints. Oracle prevents changes that might be destructive. For example, you cannot change the Column that constitutes the Primary Key unless the new Column's Nullable attribute was previously set to No.

In most cases you cannot modify an installed Table instances's constraints without dropping and replacing the Table instance during a Full or Partial Work Area installation, resulting in the loss of all data. This is not allowed in a Table instance with a validation status of Production.

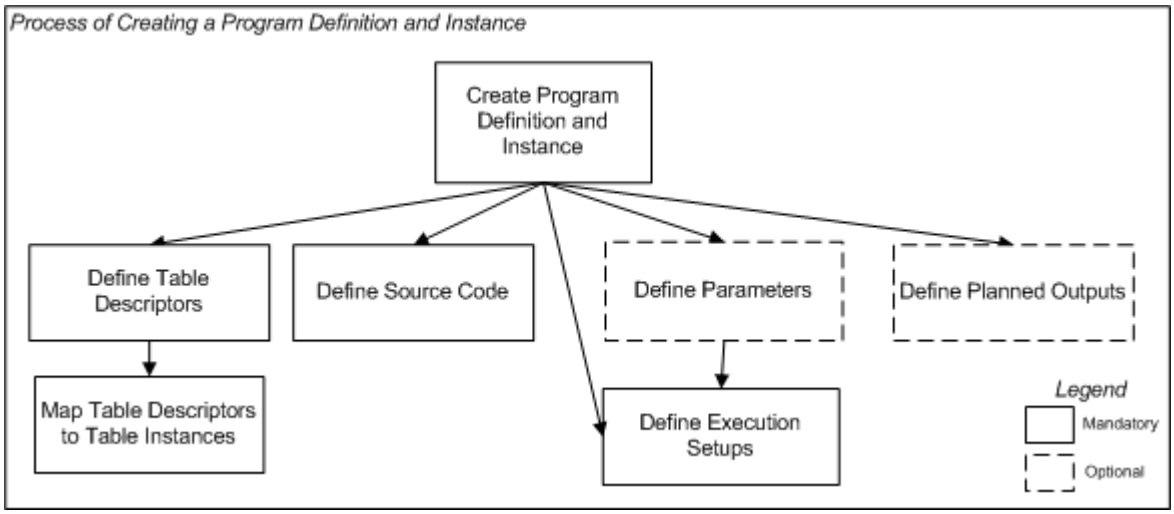
If you must modify a Production Table instance's constraints, you may be able to define a new Table instance with the required constraints and write a Program to migrate the data into it, if the data and the new constraints are compatible.

If you modify the Table definition through the instance, the system points the instance to the new version of the definition, so the changes apply to the instance as well as the definition. If you modify the definition directly in its Domain or Application Area, the changes affect only the definition, but you can upgrade instances to point to the new version.

See Defining Table Constraints and Indexes for further information.

5
Defining Programs

Figure 5-1 Process of Creating a Program Definition and Instance



This section contains information on the following topics:

- About Programs
  To create an Oracle Life Sciences Data Hub Program you write or upload source code in SAS, PL/SQL, or Oracle Reports as you normally would.
- Creating a Program
  When you create a Program in a Work Area, you are actually creating an instance of a Program definition.
- Using the Program Properties Screen
  If you are working in a Work Area, you see the properties of both the Program instance and the Program definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.
- Defining Table Descriptors
  To enable different instances of a single Program definition to run against different source or target Tables—even Tables with different names or structure—Oracle Life Sciences Data Hub requires Table Descriptors as part of the Program definition.
- Defining Source Code
- Defining Parameters
  Parameters enable you to use the same Source Code definition to achieve multiple results, controlling the processing flow differently under different conditions, or processing different data in different executions of the same Program.

- Defining Planned Outputs
  This section contains the following topics:
- Defining PL/SQL Programs
- Defining View Programs
  This section includes information on:
- Defining SAS Programs
- Defining Oracle Reports Programs
  You can use Oracle Reports as an integrated development environment to develop reports in Oracle Life Sciences Data Hub, launching Oracle Reports from a Program definition.
- Defining Informatica Programs
  Using Oracle Life Sciences Data Hub Informatica Programs, you can access Informatica tools from within Oracle Life Sciences Hub to create and edit Informatica mappings and workflows; and execute Informatica workflows from within Oracle Life Sciences Hub, on Oracle Life Sciences Hub data.
- Defining Oracle Business Intelligence Publisher Programs
  An Oracle Life Sciences Data Hub BI Publisher Program lets you use data from Oracle Life Sciences Hub Tables to create reports using Oracle BI Publisher. You can run the Oracle Life Sciences Hub BI Publisher Program from within Oracle Life Sciences Hub. The system generates a report in the desired output format(s) for Oracle Life Sciences Hub Consumers.
- Installing Program Instances
- IDE Launch Settings
  This section contains the following topics.
- Modifying Programs
  If you have the necessary privileges, you can modify a Program either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area.
- Setting Up Integrated Development Environments (IDEs)

About Programs

To create an Oracle Life Sciences Data Hub Program you write or upload source code in SAS, PL/SQL, or Oracle Reports as you normally would.

In addition, you must create a defined object in Oracle Life Sciences Hub called a Program, and create defined objects for each of the following included in your source code:

- You must define a Table Descriptor object for each source and target table or data set that your program reads from and writes to. You must then map each Table Descriptor to the actual Table instance the program reads from or writes to.
- You must define a Source Code object for the primary source code you write and for any separate subroutines, macros, or formats you use.
- You must define a Parameter object for each input, output, or input/output parameter you declare in your source code.

- You must define a Planned Output object for each output the Program will produce, including reports, error files, and log files (except SAS log files, which the system creates automatically).
- As with other Oracle Life Sciences Hub executables, you must define at least one Execution Setup to enable users to run the Program. See Creating, Modifying, and Submitting Execution Setups for further information.

Because these Program components are defined objects, Oracle Life Sciences Hub can keep them and the Program as a whole under version control, and you can validate the Program.

A Program of any technology type (SAS, PL/SQL, or Oracle Reports) can operate on any Oracle Life Sciences Hub data, regardless of the type of external system where the data originated because all Oracle LSH Tables are compatible with both Oracle tables and SAS data sets. Only one Program instance can write to any particular Table instance.

When you run a Program, the system launches the appropriate engine to execute the code, compiles source code files as necessary in that environment, and launches the primary source code file, running in batch mode. You must create an instance of a Program definition and install it and the Table instances to which it is mapped before you can run the Program; see Installing Program Instances.

Program Usage: You can use a Program in several basic ways:

- **Standalone Object**. You can use a Program to do one or both of the following:
  - generate one or more reports on data
  - manipulate data and write the transformed data to tables
- **Component of a Report Set**. Report Sets must contain Programs in order to generate the reports contained in the Report Set (see Defining Report Sets).
- **Component of a Workflow**. Any data transformation or generation of reports done within a Workflow can only be accomplished by Programs (see Defining Workflows).)
- **Container**. Source Code objects must be stored in Programs. You may want to create Programs specifically to store reusable source code files for reference as subroutines in other Programs
- **Data Viewer**. After the Program and its source Table instances are installed, you can view data in the database tables in the Integrated Development Environment (IDE). After the Program runs, you can also view data in the target Table instances.

> **Note:**
>
> For information about writing Programs that touch blinded data, see Setting and Modifying Table Attributes and "Managing Blinded Data".

Reports on Program Definitions and Instances: From the **Actions** drop-down list, you can generate reports that provide information on a Program definition or instance; see System Reports for information.

Creating a Program

When you create a Program in a Work Area, you are actually creating an instance of a Program definition.

To create a new Program instance:

1. In a Work Area, select **Program** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Program screen.
3. Choose one of the following options:
   - **Create a new Program definition and instance**. Choose this option if no Program definition exists that can meet your needs, either as it is or with some modification.
   - **Create an instance from an existing Program definition**. Choose this option if a Program definition already exists that meets your needs. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.
4. Depending on your choice, follow one of these sets of instructions:
   - Creating a New Program Definition and Instance
   - Creating an Instance of an Existing Definition

- Creating a New Program Definition and Instance
  When you select **Create a new Program definition and instance** in the Create Program screen, additional fields appear.
- Creating an Instance of an Existing Program Definition
  If you use an existing Program as a definition source, its Source Code, Table Descriptors, Parameters and other properties are already defined.

Creating a New Program Definition and Instance

When you select **Create a new Program definition and instance** in the Create Program screen, additional fields appear.

1. Enter values in the following fields:
   a. **Name**. See Naming Objects.
   b. **Description**. See Creating and Using Object Descriptions.
   c. **Program Type**. The options are: PL/SQL, Oracle Reports, SAS Program, SAS Format Catalog, SAS Macro Catalog, BI Publisher, and View. Your company may support other Program Types and they may appear in this list. Follow your company's instructions for such Program Types.
2. In the **Classification** section, select the following for both the definition and the instance:
   a. **Subtype**. Select a subtype according to your company's policies.
   b. **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Program.

The system opens the Properties screen for the new Program instance.

4. **Force Output Validation Status to 'Development'** If selected, outputs of instances of this Program definition are always created with a validation status of Development. If deselected, the outputs inherit the validation status of the Execution Setup that produced them, which in turn can inherit its validation status from the Program instance. Your company can determine the default setting using an Oracle profile; see "Setting Profile Values" in the *Oracle Life Sciences Data Hub System Administrator's Guide*. To change this value, do the following:

   a. Click the hyperlink to the Program definition in the Program in the Instance Properties section of the screen. The Program definition's Properties screen opens.

   b. Click **Update**. Fields become enterable.

   c. Select or deselect **Force Output Validation Status to 'Development'**.

   d. Click **Apply**. The system saves the change.

   e. To return to the Program instance and continue defining the Program instance and definition at the same time, click the breadcrumb link to the Program instance just above the screen title.

   > **Note:**
   >
   > This flag is a property of the Program definition, not the Program instance. Its value applies to all instances of this Program definition version.

   The setting of this attribute is version-specific; if you change it in one version, any other existing versions retain their existing value. Subsequently created versions of this Program definition get their default setting from the previous version.

   Your company can set the default value for this attribute in a lookup. For further information see "Adding and Modifying Lookup Values" in the *Oracle Life Sciences Data Hub System Administrator's Guide*. Newly created Programs get their default setting for this attribute from the lookup.

5. Define the Program details. For information and instructions see:

   a. Defining and Mapping Table Descriptors

   b. Defining Source Code

   c. Defining Parameters

   d. Defining Planned Outputs

   e. Creating, Modifying, and Submitting Execution Setups

6. Click **Check In**. The system checks in Version 1 of both the Program definition and instance.

7. Install the Program instance (see Using, Installing, and Cloning Work Areas). You can use the **Install** button on the Program instance Properties screen or install the Program instance as part of a Work Area installation. The Install button always performs an installation of type upgrade, installing the Program instance only if the current version has not been installed previously.

8. Validate both the definition and the instance according to your company's policies.

   For information on creating the different types of Programs, see:

   - Defining PL/SQL Programs.
   - Defining SAS Programs.
   - Defining Oracle Reports Programs.
   - Defining Oracle Business Intelligence Publisher Programs

Creating an Instance of an Existing Program Definition

If you use an existing Program as a definition source, its Source Code, Table Descriptors, Parameters and other properties are already defined.

See Creating an Instance of an Existing Definition for instructions.

After you have created the Program instance, you must map the Table Descriptors to Table instances; see Mapping Table Descriptors to Table Instances . You must also create at least one Execution Setup for the Program instance; see Creating, Modifying, and Submitting Execution Setups.

Using the Program Properties Screen

If you are working in a Work Area, you see the properties of both the Program instance and the Program definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See Modifying Programs for information on modifying Programs.

See also Figure 5-1.

This section contains the following topics:

- Instance Properties
  You can see the following instance properties:
- Definition Properties
- Buttons
  From a Program instance in a Work Area, you can use the following buttons:

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Definition:** This field specifies the Program definition to which this Program instance points.
  To upgrade to a new version of the same definition, use the **Upgrade to Latest** button. See Upgrading to a Different Definition Version from an Instance.
- **Blind Break:** This field indicates whether you can see real or dummy data in blinded Table instances when you work on this Program from an IDE. Click

Launch Settings to make this selection. The choices depend on your privileges. If none of the Table instances mapped to the Program instance contains either blinded or unblinded data, the only possible setting is Not Applicable. See IDE Launch Settings for further information.

- **Shared Snapshot Label:** The default value you see here comes from the default Execution Setup for this Program, if there is one. Otherwise the default value is determined by your privileges. If the source Table instances have shared snapshot labels, you can click **Launch Settings** and select one of them. See IDE Launch Settings for further information.
- **Validation Status:** This field displays the current validation status of the Program instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Program: Installable or Non Installable. Programs have an additional status called Installable IDE. You can install a Program without a Source Code, if the Program has the Installable IDE status. You can work on such a Program's Source Code in an IDE. See Installation Requirements for Each Object Type.
- **Version:** This field displays the current version number of the Program instance.
- **Version Label:** This field displays the version label, if any, for the current Program instance version.
  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.

Definition Properties

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to modify Table Descriptors, Source Code, Parameters, or Planned Outputs. However, you can change Table Descriptor mappings without checking out the definition. See Understanding Object Versions and Checkin/Checkout for further information.
- **Latest Version:** If set to **Yes**, this Program instance is pointing to the latest version of the Program definition. If set to **No**, this Program instance is pointing to an older version of the Program definition.
- **View Latest:** You can see this button only if the current Program instance does not point to the latest definition version. Click this button to view the latest Program definition.
- **Upgrade to Latest:** This button is grayed out if the current Program instance already points to the latest Program definition. Click this button to upgrade the current Program instance to the latest definition version. For more information on upgrading instances, see Upgrading Object Instances to a New Definition Version.
- **Checked Out By:** This field displays the username of the person who has the Program definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Program Type:** This field displays this Program definition's type: Oracle Reports, PLSQL, SAS Program, SAS Format Catalog, or SAS Macro Catalog. See Defining

PL/SQL Programs, Defining SAS Programs, and Defining Oracle Reports Programs.

- **Development Tool:** This field displays the tool required to work on the source code of the Program: Oracle Reports, PL/SQL, or SAS.
- **Validation Status:** This field displays the current validation status of the Program definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Program in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Program: Installable or Non Installable. See Installation Requirements for Each Object Type.

Buttons

From a Program instance in a Work Area, you can use the following buttons:

- **Install:** Click **Install** to install the Program instance, including mapping target Table Descriptors and installing mapped target Table instances; see Installing Program Instances. For a list of reasons a Program instance may not be installable, see Installation Requirements for Each Object Type.
- **Launch IDE:** Click **Launch IDE** to launch the integrated development environment (IDE) in which you write your program source code.
- **Submit:** Click **Submit** to run the Program instance. Before you can run the Program, you must install it and create an Execution Setup for it (select **Execution Setups** from the **Actions** drop-down list).
- **Update:** Click **Update** to modify the Program instance properties. See Modifying Program Instance Properties.
- **Launch Settings:** Click **Launch Settings** to set the blinding status and currency of the data you want to view while developing your Program; see IDE Launch Settings.
- **Check In/Out and Uncheck:** Click these buttons to check out, check in, or uncheck the Program definition. Different buttons are displayed in the Program Definition Properties section depending on the Checked Out Status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of the person who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.
- **View Latest/Upgrade to Latest:** If the definition is not the latest version, you can click to view the latest version and upgrade to the latest version if you want to. See Upgrading to a Different Definition Version from an Instance.

Defining Table Descriptors

To enable different instances of a single Program definition to run against different source or target Tables—even Tables with different names or structure—Oracle Life Sciences Data Hub requires Table Descriptors as part of the Program definition.

You must include one Table Descriptor for each Table instance the Program will read from or write to. Like a Table instance, a Table Descriptor contains a pointer to a Table definition. The difference is that a Table Descriptor exists only inside a Program or other executable object definition, while a Table instance is installed independently in the database. See Defining and Mapping Table Descriptors for further information.

You must map each Table Descriptor to the corresponding Table instance that the Program instance will read from or write to. The system can do the mapping automatically if you choose to create Table Descriptors from existing Table instances or if the Table Descriptor has the same name as the Table instance. However, if the Table instance is different enough from the Table Descriptor, you must map them manually. See Mapping Table Descriptors to Table Instances).

The Program definition's source code refers to the Table Descriptor by name and Column name as if it were an actual table or SAS data set containing data. The system uses the mappings to translate the names used in the source code to those of the Table instance to which the Table Descriptor is mapped.

> **Note:**
>
> Only one Program can write to any particular Table instance. The system prevents you from mapping a Table instance to more than one target Table Descriptor.

Target Table instances must have a processing type. Be sure that the processing type of each target Table instance is compatible with your source code; see Data Processing Types for further information.

There are several ways to create a Table Descriptor:

- Creating Table Descriptors from Table Instances and Simultaneously Mapping Them
- Adding a New Target Table Descriptor
- Adding Target Table Descriptors from a SAS File
- Adding a Target Table Descriptor from an Existing Table Definition

See also Mapping Table Descriptors to Table Instances .

For information on how you can make data available to Program instances in an Integrated Development Environment (IDE), see IDE Launch Settings.

Defining Source Code

For related information, see also:

- Defining PL/SQL Programs
- Defining SAS Programs
- Defining Oracle Reports Programs
- Defining Oracle Business Intelligence Publisher Programs

This section contains information about Source Code in Oracle LSH, including:

- About Source Code
- Creating Source Code
  When you create Source Code in a Program, you can either create an instance of an existing Source Code definition, or a new Source Code definition and instance at the same time:
- Calling APIs from Source Code
  Oracle publishes public PL/SQL APIs that allow you to do many things programmatically that you can do through the user interface, including creating, modifying, and installing objects.
- Creating and Using Static Reference Source Code
  This section contains the following topics:
- Upgrading Source Code And Undoing Source Code Upgrades
  This section contains the following topics:

About Source Code

A Source Code definition encapsulates the file containing the actual source code, so that the source code is stored under version control, in compliance with industry regulations (see Versions of Component Objects).

Every Oracle LSH Program must contain one **primary** source code object and may contain any number of additional (**secondary**) source code objects serving as subroutines.

> **Note:**
>
> The Source Code for Oracle Life Sciences Data Hub Programs of the BI Publisher adapter type is automatically generated by the system. Do not edit or upload the Source Code manually or the BI Publisher Program may not work properly.
>
> See About Oracle BI Publisher Program Source Code

Primary Source Code: Each Program contains one primary Source Code definition, listed first and given an Order number of one (1). When you execute the Program, the system launches the Source Code definition you have defined as Primary. Normally you write the primary source code especially for a particular Program. The primary Source Code definition contains the file the system executes when the Program is submitted. The primary source code references Table Descriptors, Parameters, Planned Outputs, and other Source Code instances defined in the Program by their Oracle or SAS name, as appropriate for the technology type (see Writing Primary Source Code in PL/SQL and SAS Program and Source Code Types).

Secondary Source Code: Secondary Source Code objects are those with any order number other than one (1). They are SAS macros or formats or PL/SQL packages that are, in most cases, called by the primary source code.

Source Code definitions have a Sharable attribute that, if set to **Yes**, makes them available for use as a definition source for Source Code instances in other Programs. You can create Program definitions especially for the purpose of storing sharable Source Code definitions. If you are working in SAS, you must create a Program

definition of type Macro Catalog to store macros, or a Program definition of type Format Catalog to store formats. If you are working in Oracle technologies (Oracle Reports or PL/SQL) you can create a Program definition of type PL/SQL to hold sharable PL/SQL packages.

You can also create secondary Source Code especially for a particular program. In this case Oracle Life Sciences Data Hub stores both the Source Code instance and its definition in the Program in which you create them.

See Upgrading Source Code And Undoing Source Code Upgrades for information on how to upgrade Source Code instances pointing to a sharable Source Code definition.

For further information, see:

- Creating a SAS Macro Catalog
- Creating a SAS Format Catalog
- Creating a PL/SQL Package Storage Program
- Execution
  When you submit an Execution Setup to run a Program, the system does the following in sequence:

Execution

When you submit an Execution Setup to run a Program, the system does the following in sequence:

1. Initializes the batch environment for the appropriate technology
2. If necessary, compiles all the Source Code files in the order in which they are listed in the Source Code subtab (all SAS macros and formats included in the Program are compiled)
3. Launches the primary Source Code file using the Parameter values specified in the Execution Setup
1.
   Note:

   You cannot set a Static-reference Source Code as primary. Also, you cannot set a Source Code that points directly to a Program definition as primary.
4. Launches any secondary Source Code files when they are called from the primary file
5. Writes data to Table instances, if so directed by the source code
6. If directed by the source code, generates reports and classifies them as specified in Planned Output definitions

Creating Source Code

When you create Source Code in a Program, you can either create an instance of an existing Source Code definition, or a new Source Code definition and instance at the same time:

1. In the Source Code subtab of a Program, click **Add**. The system displays the Create Source Code screen.
2. Choose one of the following options:
   - **Create a new Source Code definition and instance**. Select this option in any of the following situations:
     - You are creating the primary Source Code for a Program
     - The source code you want to use exists only outside Oracle Life Sciences Data Hub (you can upload the actual source code file, but you must create a new Source Code definition to store the uploaded file)
     - You are creating a subroutine that does not yet exist in Oracle Life Sciences Hub (see Creating a SAS Macro Catalog, Creating a SAS Format Catalog, and Creating a PL/SQL Package Storage Program).
   - **Create an instance of an existing Source Code definition**. Select this option in any of the following situations:
     - You are working in a PL/SQL Program and you want to reference a package in another Program.
     - You are working in a SAS Program and you want to reference a macro or format in another Program.

     Your company may have Programs created especially to store sharable Source Code files in a designated Library or Application Area.

     For more information on using SAS Format Catalogs and SAS Macro Catalogs, see Using a SAS Macro Catalog and Using a SAS Format Catalog.
3. Depending on your choice, follow one of the following sets of instructions:
   - Creating a New Source Code Definition and Instance
   - Creating an Instance of an Existing Source Code Definition

You can also create a source code instance that points to a Program definition. This is so that you can use the same SAS Macro Catalog and SAS Format Catalog in other Programs. See SAS Program and Source Code Types.

- Creating a New Source Code Definition and Instance
  When you select **Create a new Source Code definition and instance** in the Create Source Code screen, additional fields appear.
- Creating an Instance of an Existing Source Code Definition
  If you use an existing Source Code as a definition source, its contents are already defined.

Creating a New Source Code Definition and Instance

When you select **Create a new Source Code definition and instance** in the Create Source Code screen, additional fields appear.

This section contains instructions for all technology types. For instructions specific to each technology, see the following sections:

- Defining PL/SQL Programs
- Defining SAS Programs
- Defining Oracle Reports Programs

- Defining Oracle Business Intelligence Publisher Programs

> **Note:**
>
> Oracle Life Sciences Data Hub creates the Source Code automatically for Oracle Life Sciences Hub BI Publisher Programs. Do not create or edit this Source Code. See About Oracle BI Publisher Program Source Code for more information.

1. Enter values in the following fields:
   - **Name**. The system uses the name you enter for the actual source code file. If you do not specify an extension, Oracle Life Sciences Hub appends the default extension for that technology type: .sas for SAS source code, .rdf for Oracle Reports, or .sql for PL/SQL. Do not use reserved words or special characters. See Naming Objects for further information.
   - **Description**. See Creating and Using Object Descriptions.
   - **File Type**. Select the file type from the list. The choices vary depending on the Program Type. For an explanation, see Defining SAS Programs or Defining Oracle Reports Programs. PL/SQL Programs' Source Code must have a SQL file type.
   - **Sharable**. Select **Yes** to make the Source Code definition available for reuse. See Creating a PL/SQL Package Storage Program and Creating a SAS Macro Catalog for information about sharable Source Code definitions.
     Depending on your company's validation policies you may choose not to set the **Sharable** flag to **Yes** until you have fully tested and validated the Source Code. You can change the setting at any time in the Source Code subtab by selecting the Source Code and clicking either **Set Sharable** or **Set Not Sharable**.

> **Note:**
>
> You cannot set any SAS Macro Catalog or SAS Format Catalog Source Code instance that points directly to a Program definition as Sharable.

2. Enter additional fields specific to each Program type:
   - **SAS File Reference Name**. If the Program is of type SAS, you may need to enter a SAS File Reference Name:
     - The first Source Code you create of type Program is automatically created as the Program's primary Source Code. The SAS File Reference Name value defaults to `MAINPRG` and you cannot change it. If you later set another Source Code as primary, the system automatically changes its SAS File Reference Name to MAINPRG and changes the original primary Source Code's SAS File Reference Name to the Source Code instance name, truncated to 8 characters.
     - If the Source Code is a secondary Source Code that is not shared from another Source Code, the value defaults to the Source Code's name, truncated to 8 characters. You can change this value.

- If the Source Code is shared from another Program, the SAS File Reference Name defaults from the shared Source Code and you cannot change it.
- **Note:**

  In cases where the SAS File Reference Name value defaults, if there are two names with the same value, the system truncates the second one by one character and appends 1 (or increments the number if there are three or more).

- **Oracle Package Name**. If the Program is of type PL/SQL, enter a package name. The package name must be unique within the Program. The package name must match the package name in the actual PL/SQL source code.
- **Oracle Procedure Name**. If the Program is of type PL/SQL, enter a procedure name. The procedure name must match the procedure name in the actual PL/SQL source code.

Note:

If the Program is of type PL/SQL and you are defining the primary Source Code, you must enter both a package name and a procedure name.

3. Write or upload the actual source code. Do one of the following:
   - For text-based source code files (PL/SQL and SAS), write code directly in the Source Code Editor box.
   - If a source code file that fits your needs already exists on your PC or network, click the **Upload** button to look for and select the file.

Note:

For Informatica Programs, you must only upload existing XML files that contain Informatica mappings and workflows.

   - Click **Launch IDE** to open the development environment. Write source code there to use in the Program, then upload the source code file to the Source Code definition in Oracle Life Sciences Hub.

Note:

Before you can use the **Launch IDE** button you must install the Program. To make the Program installable you must either create and map at least one Table Descriptor or add one Source Code object (Steps 1 and 2 above). For more information see Installing Program Instances.

> **Note:**
>
> If you are using SAS in a connected mode and the SAS development environment does not open when you click **Launch IDE**, you may need to define services for the environment and start Oracle LSH Distributed Processes Server code in the same location. See "Stopping and Starting Services and Queues" in the *Oracle Life Sciences Data Hub System Administrator's Guide*.
> If you are using SAS in Disconnected mode, SAS does not open when you click Launch IDE. Instead, Oracle LSH puts the required data set files on your personal computer and displays a message giving the location. You must then open SAS locally and access the files as necessary.
>
> See Connecting to SAS for further information.

4. Click **Save and Continue** to save your work. The system saves your work in the database and returns to the Program instance screen.

   By default the system sets the first Source Code you create for a Program to Primary. If you want to specify that a different Source Code is the primary one—the one executed first—in the Source Code subtab check the Select check box of the Source Code you want to set as Primary and click **Set As Primary**. The system changes the Primary setting for that Source Code to Yes, and for the previous primary Source Code to No.

5. Validate both the definition and the instance according to your company's policies.

Creating an Instance of an Existing Source Code Definition

If you use an existing Source Code as a definition source, its contents are already defined.

However, you must specify whether or not to use a static reference; see Creating and Using Static Reference Source Code for information.

See Creating an Instance of an Existing Definition for general instructions.

> **Note:**
>
> If the Program containing the Source Code or the Source Code itself is ever deleted, the Programs that contain instances of the deleted Source Code will no longer compile.

Calling APIs from Source Code

Oracle publishes public PL/SQL APIs that allow you to do many things programmatically that you can do through the user interface, including creating, modifying, and installing objects.

You can call these APIs within the context of defined Oracle Life Sciences Data Hub Programs without any extra security. Refer to the *Oracle Life Sciences Data Hub System Application Programming Interface Guide* for details.

For an example of calling a PL/SQL API from a SAS Program, see Calling an API to Capture Output Parameter Values.

Creating and Using Static Reference Source Code

This section contains the following topics:

- About Static Reference Source Code
  You can write source code (PL/SQL package or SAS format) that is intended for reuse in other Programs.
- Creating a Source Code for Use as a Static Reference
  If you are creating a PL/SQL Program or SAS Format Catalog to store static reference Source Code definitions, you must create an instance of the Program and map one of its source Table Descriptors to each Table instance required to be read, and install both the Program instance and the Table instance.
- Using Static Reference Source Code
  In the Program in which you want to use the Source Code:

About Static Reference Source Code

You can write source code (PL/SQL package or SAS format) that is intended for reuse in other Programs.

If that Source Code is located in an installed Program instance—usually because it reads data such as lookup values from an Oracle Life Sciences Data Hub Table instance—you must specify a static reference when you create an instance of the Source Code in another Program.

For example, if you want to supply an Investigator ID and return an Investigator Name, you can do either of the following:

- Create a Table Descriptor in your Program and map it to a Table instance that includes columns for both Investigator ID and Name and reference that Table Descriptor in the Source Code you create for your Program. This method does not involve a static reference Source Code.
- Reference a sharable Source Code in another Program instance whose purpose is to take an Investigator ID and return the corresponding Investigator Name. The Program instance that contains this sharable Source Code must have a Table Descriptor that is mapped to a Table instance that includes columns for both Investigator ID and Name. This is a static reference Source Code.

> **Note:**
>
> Oracle Life Sciences Hub does not support creating a static reference to Source Code contained in a Program instance inside a Report Set.

Creating a Source Code for Use as a Static Reference

If you are creating a PL/SQL Program or SAS Format Catalog to store static reference Source Code definitions, you must create an instance of the Program and map one of its source Table Descriptors to each Table instance required to be read, and install both the Program instance and the Table instance.

Do not define snapshot labels for these Table instances and do not set their Blinding Flag to Yes.

Using Static Reference Source Code

In the Program in which you want to use the Source Code:

1. Add a Source Code object by creating an instance of an existing Source Code definition. The Static Reference attribute appears.
2. Select **Yes** for Static Reference.
3. Click the Search icon for the Definition Source field.

   The system then searches for installed Program instances rather than Program definitions, because the Program containing the static reference Source Code must be mapped to the necessary installed Table instance. Also, because only one version of a Program instance can be installed at any given time, your Program always references the current installed version of the static reference Source Code's Program instance.
4. Enter the Domain, Application Area, and Work Area where the Program instance is located and select the Program instance.
5. Apply.

Upgrading Source Code And Undoing Source Code Upgrades

This section contains the following topics:

If the Source Code definition your Source Code instance is pointing to is not the most current version, Oracle LSH sets the **Latest Version** field to **No** and allows you to view the latest version and upgrade to it if you want to; see Upgrading a Single Source Code Instance .

> **Note:**
>
> You cannot upgrade a Source Code instance whose definition is statically referenced. See Creating and Using Static Reference Source Code.

You can also undo an upgrade or choose a noncurrent version of a Source Code definition at any time; see Undoing Source Code Instance Upgrades.

Working in the Source Code definition, you can upgrade all instances to the latest version, using the **Upgrade All Instances** button or the **Actions** drop-down list.

If you created a Source Code instance along with a definition, then your Source Code definition and instance are synchronized unless at some point you or someone else manually pointed the Source Code instance to another version of the definition; see Undoing Source Code Instance Upgrades.

If you created a Source Code instance that points to an existing Source Code definition, any changes that have been made to the Source Code definition after you created the Source Code instance are not reflected in the instance and **Latest Version** is set to **No**.

For Source Code instances that point to an existing Source Code definition, you can see a hyperlink to the Source Code definition that the Source Code instance points to, in the instance properties section of the Source Code screen.

> **Note:**
>
> A Source Code instance and the Source Code definition it points to, both always share the same screen. Therefore, in this section **Source Code screen** refers to the screen that shows Source Code instance properties in the upper portion and Source Code definition properties in the lower portion of the screen.

- Upgrading a Single Source Code Instance
  You can upgrade a single Source Code instance in one of the following ways:
- Upgrading Multiple Source Code Instances
  A Source Code instance may point to a Source Code definition or, in the case of SAS Source Code instances, to a SAS Macro or Format Catalog Program definition.
- Undoing Source Code Instance Upgrades
  You undo a Source Code instance upgrade by pointing the Source Code instance to an earlier version of the Source Code definition than the one you upgraded the Source Code instance to.

Upgrading a Single Source Code Instance

You can upgrade a single Source Code instance in one of the following ways:

- Upgrade to Latest button. This method allows upgrade to the latest version of the Source Code definition only.
- Upgrade Instance from the **Actions** drop-down list. This method allows changing to any version of the Source Code definition.
- Upgrade to Latest
  Use the **Upgrade to Latest** button to upgrade the Source Code instance to the latest version of its definition.
- Upgrade Instance
  Use the **Upgrade Instance** item from the **Actions** drop-down list on the Source Code screen, to upgrade the Source Code instance to any version of its definition.

Upgrade to Latest

Use the **Upgrade to Latest** button to upgrade the Source Code instance to the latest version of its definition.

- The Source Code definition to which you want to upgrade is not checked in.
- The Program definition containing the Source Code instance you want to upgrade is not checked out, or is checked out by someone else.
- The Source Code instance is already pointing to the latest version of its Source Code definition (**Latest Version** is set to **Yes**).

- The Source Code Definition is a static reference Source Code; see Creating and Using Static Reference Source Code.
- You do not have Modify privileges on the Source Code instance

To upgrade a Source Code instance to the latest version of its definition using the Upgrade to Latest button, do the following:

1. Navigate to the Program instance or definition that contains the Source Code instance you want to upgrade.
2. In the Source Code subtab, click the Source Code's hyperlink in the Name column. The Source Code screen opens.
3. If you want to look at the latest version of the Source Code definition before upgrading, click **View Latest** in the Definition Properties section of the screen.

   To upgrade, click **Upgrade to Latest**. You receive a confirmation message.
4. In the confirmation message, click **Yes**. The system upgrades the Source Code instance to the latest version of its source definition.

Upgrade Instance

Use the **Upgrade Instance** item from the **Actions** drop-down list on the Source Code screen, to upgrade the Source Code instance to any version of its definition.

This option is not available (the **Go** button is grayed out) if:

- The Source Code definition to which you want to upgrade is not checked in.
- The Program definition containing the Source Code instance you want to upgrade is not checked out, or is checked out by someone else.
- he Source Code Definition is a static reference Source Code; see Creating and Using Static Reference Source Code.
- You do not have Modify privileges on the Source Code instance.

To point a Source Code instance to any version of its definition, do the following.

1. Navigate to the Program instance or definition that contains the Source Code instance you want to upgrade.
2. In the Source Code subtab, click the Source Code's hyperlink in the Name column. The Source Code screen opens.
3. If you want to look at the latest version of the Source Code definition before upgrading, click**View Latest** in the Definition Properties section of the screen.

   To change the underlying definition to a different version, select **Upgrade Instance** from the **Actions** drop-down list.
4. Click **Go**. The system displays the available versions of the Source Code definition in the lower portion of the screen.

   For each version of the definition, the system displays the following information:

   - **Source Code Version**. The version number of the Source Code definition version.
   - **Upgrade**. The **Upgrade** icon is grayed out if the current Source Code instance already points to that Source code definition version, or if that Source Code definition version is currently checked out (someone is currently modifying it).

> **Note:**
>
> If someone is currently creating a new version, the new version is not displayed at all.

- **Program Name**. The name of the Program definition that owns the Source Code definition. This remains the same for one Source Code definition.
- **Program Version**. The version number of the Program that owns that Source Code definition version.
- **Program Validation Status**. The validation status of the Program that owns that Source Code definition version.
- **Program Version Label**. The label associated with the Program definition version, if any.
- **Program Checked Out By**. If the Program definition version is checked out, the system displays the username of the person who checked it out. You cannot upgrade to a version that is checked out, and only the person who checked it out can check it in.

> **Note:**
>
> People with Checkin Administrator privileges can check in objects checked out by other users.

5. Click the icon in the **Upgrade** column for the version to which you want to point the instance.

   The system upgrades the Source Code instance and grays out its row, as it now points to the version you selected.

Upgrading Multiple Source Code Instances

A Source Code instance may point to a Source Code definition or, in the case of SAS Source Code instances, to a SAS Macro or Format Catalog Program definition.

You can upgrade Source Code instances from a Source Code screen or in the case of SAS Source Code instances, from a Program instance's Properties screen.

As with other object types, you can go to a version of a definition and upgrade all the instances that point to it.

If the Source Code definition is a SAS Macro or Format Catalog, follow the instructions for upgrading Program and other object definitions at Upgrading One or More Instances from the Definition.

- From the Source Code definition screen
  To upgrade one or more Source Code instances from a Source Code definition, do the following:
- From the Program Definition screen
  To upgrade multiple Source Code instances that refer to a Program definition, do the following:

From the Source Code definition screen

To upgrade one or more Source Code instances from a Source Code definition, do the following:

1. Navigate to the Program that contains the Source Code definition whose instances you want to upgrade.
2. Check in the Program definition if it is not already checked in.
3. In the Source Code subtab, click the Source Code's hyperlink in the Name column. The Source Code screen opens.
4. Click the **Upgrade All Instances** button in the Source Code definition properties section of the screen.
5. The system opens the Upgrade Instances screen with all instances of the Source Code definition displayed.

> **Note:**
>
> If the check box next to an instance is grayed out, then either the Program instance that owns it is checked out by someone else, or the Source Code instance already points to the latest Source Code definition.

1. For each instance, the system displays the following information:
   - **Program Name**. The Program name that contains the Source Code instance.
   - **Program Version**. The version number of the Program that contains the Source Code instance.
   - **Program Version Label**. The version label of the Program that contains the Source Code instance.
   - **Program Validation Status**. The validation status of the Program that contains the Source Code instance.
   - **Program Checked Out By**. The name of the person who has checked out the Program definition, parent to the Source Code instance. If a person other than you has the Program checked out, then you cannot upgrade the Source Code instance: the check box next to it is grayed out.
   - **Source Code Definition Version**. The version number of the Source Code definition to which that instance currently points.
   - **Source Code Name**. The name of the Source Code instance to which that instance currently points.
   - **SAS File Reference Name**. The SAS file reference name, if any.
   - **Container**. The Domain > Application Area hierarchy for the instance.
6. Select one or more instances to upgrade. You can use the **Select All** and **Select None** functions and/or select or deselect instances individually by checking or unchecking their **Select** checkbox. Instances that already point to the current version of the definition cannot be selected.

7. Click **Upgrade**. The system changes the source definition of the selected instances to the version of the definition where you are working.

> **Note:**
>
> If the **Upgrade** button is not enabled, then the Source Code definition you want to upgrade to, is not checked in. To check in that Source Code definition, you must check in the Program definition that owns it.

> **Note:**
>
> The newly upgraded version of the definition is not necessarily the latest version. It is the version you are currently working on. To go to the latest Source Code definition version, click the **View Latest** button from the Source Code definition properties section of the Source Code screen.

From the Program Definition screen

To upgrade multiple Source Code instances that refer to a Program definition, do the following:

1. Navigate to the Program definition in the Applications tab.
2. Check in the definition if it is not already checked in.
3. From the **Actions** drop-down list, select **Upgrade All Instances** and click **Go**. The system opens the Upgrade Instances screen displaying all Program instances and Source Code instances that point to the Program definition.
4. From the **View** drop-down list, select **Program Definition (Source Code Instances)**. The system refreshes the screen and lists only the Source Code instances that point to this Program definition.
5. For each instance, the system displays the following information:

> **Note:**
>
> In the following, **Object** refers to the Program definition that owns the Source Code instance. If you are viewing this information with the selection **Program Instance** or **Both** in the **View** drop-down list (see point 4 above), then **Object** refers to a Program instance, where it is a Program instance that is referring to this Program definition.

- **Object Name**. The name of the Program definition that contains the Source Code instance.
- **Object Type**. The type of the object. See Note above.
- **Object Version**. The version number of the Program definition that contains the Source Code instance.
- **Version Label**. The version label of the Program definition that contains the Source Code instance.

- **Installed Version**. This field is not applicable for Source Code instance upgrades because a Source Code instance is referred to by its owning Program definition and Oracle LSH object definitions cannot be installed. This field refers to the most recent version of the Program instance that was successfully installed. It is relevant only for Program instances pointing to this Program definition.
- **Validation Status**. The Program definition's validation status.
- **Checked Out By**. The name of the person who has checked out the Program definition, parent to the Source Code instance. If a person other than you has the Program checked out, then you cannot upgrade the Source Code instance that points to it: the check box next to it is grayed out.
- **Definition Version**. The version number of the definition to which the instance currently points.

> **Note:**
>
> In the following, **Parent** refers to the **Object's** parent. So for Source Code instances, it is the parent of the Program definition that owns the Source Code. That is, parent of a Program definition - such as an Application Area.

- **Definition Validation Status**. This field does not apply to Source Code instances. It is the validation status of the Program instance that points to a version of the current Program definition.
- **Parent Name**. The name of the Parent object.
- **Parent Object Type**. The type of object that contains the Program definition that owns the Source Code instance.
- **Parent Status**. The parent's installation status.
- **Parent Validation Status**. The parent's validation status.
- **Source Code Name**. The name of the Source Code instance.
- **SAS File Reference Name**. The SAS file reference name, if any.
- **Container**. The Domain > Application Area hierarchy for the Program definition that owns the Source Code instance.

6. Select one or more instances you want to upgrade. You can use the **Select All** and **Select None** functions and/or select or deselect instances individually by checking or unchecking their **Select** checkbox. Instances that already point to the current version of the definition cannot be selected.
7. Click **Upgrade**. The system changes the source definition of the selected instances to the new version of the definition.

> **Note:**
>
> When upgrading a Source Code instance, you are actually upgrading the Program definition that owns the Source Code instance. However, for the sake of readability, the document describes operations on the Source Code instance. Please remember that you cannot check in or check out a Source Code instance, you can perform these operations on only the Program definition that owns it.

Note the following:

- The Upgrade button is grayed out if the Program definition you want to upgrade to, is not checked in. Click **Return** to go back to the previous screen and check in the definition first.
- If a version of the Source Code Instance already points to the Program definition you want to upgrade it to, then its row is grayed out.
- If the latest version of the Source Code instance is checked out, then all older versions are grayed out.

  If you want to upgrade an older version of the Source Code instance, then you should check in the latest Program definition that owns the Source Code instance and return to this screen. Older version are now available for upgrading. When you select an older version and click **Upgrade**, the system checks out the Source Code instance, creates a new version and then upgrades it. The system refreshes the screen with this information.

Undoing Source Code Instance Upgrades

You undo a Source Code instance upgrade by pointing the Source Code instance to an earlier version of the Source Code definition than the one you upgraded the Source Code instance to.

You can undo Source Code upgrades for a single Source Code instance or for multiple Source Code instances.

Single Source Code instance: To undo a Source Code instance upgrade, see Upgrade Instance.

Multiple Source Code instances: To undo multiple Source Code instances, see Upgrading Multiple Source Code Instances.

Defining Parameters

Parameters enable you to use the same Source Code definition to achieve multiple results, controlling the processing flow differently under different conditions, or processing different data in different executions of the same Program.

For example, you can use the same Program to process data for different studies by defining a Parameter for the study name.

If you use a parameter in your source code internally only, you do not need to create a Parameter definition for it. However, if you want the Parameter to be settable at runtime or in the Execution Setup definition, you must define it. When you define a

Parameter you can give it a default value and/or a list of values. See Defining Parameters for further information.

> **Note:**
>
> Programs of BI Publisher adapter type contain predefined Parameters that you should not modify. See Setting Oracle BI Publisher Program Parameters.

Defining Planned Outputs

This section contains the following topics:

- About Planned Outputs
  A Planned Output is a placeholder for a file to be generated by a Program during execution.
- Defining a Planned Output

About Planned Outputs

A Planned Output is a placeholder for a file to be generated by a Program during execution.

There are three types: Primary, Secondary, and Error File. You can define any number of any type of Planned Output.

Planned Outputs are not required; you can create Programs whose purpose is to transform and write data to a Table rather than to produce a report.

You must create a different Planned Output for each file you want to generate. If you want to create the same report in two file types, define a Planned Output for each of them (in that case you might also want to use a Parameter to determine which one to produce at runtime).

The source code for the Program must specify how to create each Planned Output defined in the Program and refer to each one by the name appropriate to the development environment (for example, Oracle or SAS).

Primary Output: A primary output is a report on data generated during the successful execution of the Program. The purpose of the Program is to produce one or more primary outputs (and may also transform data). You must write source code that produces the report you want, and refer to the primary Planned Output by name in the source code.

You can define more than one primary output for a single Program. For example, you could create two primary Planned Outputs to present the same information in two ways, such as a table and a graph; or you could divide the data results into two or more categories for presentation, with a Planned Output for each category.

Secondary Output: A secondary output is one that is not defined as either Primary or Error File; for example, a log file. The execution engine produces a log file for every execution of an Oracle Life Sciences Data Hub Program, but you may or may not need to define a Planned Output as a placeholder for the log file:

- The system automatically creates one secondary Planned Output for each SAS-type Source Code definition, to hold the log file. SAS log files therefore appear in the Reports tab navigation tree.
- You can view the log file for Programs of all Oracle types from the Job screen. If you want to have the log file appear as an entry in the Reports tab navigation tree as well, you must create a secondary Planned Output for it.
- For Oracle Life Sciences Hub Informatica Programs, defining Planned Outputs for the log files has no effect. You can see the log files through the job. The Oracle Life Sciences Hub Informatica Program generates an unexpected output for each log file at the time of execution, but does not fail on account of this.

Error File: An Error File Planned Output is a file generated automatically by the system if a Program execution fails to generate a primary output defined as Required. To define an Error File Planned Output you must also define at least one primary Planned Output as Required in the same Program definition.

The very existence of an error file as an output indicates a failure.

Classifying Planned Outputs: You can classify Planned Outputs. The system assigns the same classification values to the actual output when the Program generates it. Users find report outputs by their classifications in the Reports tab of the Oracle Life Sciences Hub user interface. See Defining Planned Outputs and Classifying Outputs.

Report Sets and Planned Outputs: To use a Planned Output as a report in a Report Set, you must create an instance of the Program that generates the Planned Output in a Report Set Entry in the Report Set.

All the primary Planned Outputs generated by the Program are included in the Report Set Entry, in the order in which they are displayed in the Program. If there are more than one primary Planned Outputs and you want to be able to choose which one(s) to include, you should create the Program with an input Parameter for this purpose.

Secondary outputs and error files are not included in the Report Set's table of contents.

The Report Set produces the output in the file type you specify as part of the Planned Output definition. If you have Publishing Light installed and want to use those features for this Report, you must specify a file type of PDF.

Defining a Planned Output

To define a Planned Output:

1. In the Planned Output subtab of a Program, click **Add**. The system displays the Planned Output For screen.
2. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Title**. Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field.
   - **Description**. See Creating and Using Object Descriptions.
   - **File Name**. You must define a File Name for each defined Planned Output of a Program of any type. For most technology types it must include a file extension. The system converts any spaces you enter to underscores (_). The

system uses the File Name to match the actual generated output to the corresponding Planned Output in order to classify the actual output file.

In a non-SAS Program, refer to the output as the File Name in your source code. In a SAS Program, make the File Name the same as the File Reference Name plus a file extension; for example, if the File Reference Name is `out1`, make the File Name `out1.pdf`.

- **Primary**. If **Yes**, indicates that the output file will contain a report on data generated by the Program during execution.
- **Error if generated**. If **Yes**, the system generates an error if Program execution fails to generate the output.
- **Error if not generated**. If **Yes**, the system generates this output only if Program execution fails to generate a Primary Planned Output. Its presence is an indication that the Program failed. You must write the source code to generate the text of the file. Note: You cannot define an Error file as Required.

> **Note:**
>
> You do not need to define a Planned Output for the .log file.

- **File Reference Name**. You must define a File Reference Name for each Planned Output of a SAS Program. It should conform to SAS rules. You should also define a File Reference Name if you plan to use an instance of the Program in a Report Set and to pass Report Set Entry properties' values to the Program. Oracle recommends making Planned Output File Reference Names unique within a Program, but this is not enforced. See Passing Report Set Entry Values to and from Programs.

  The File Reference Name defaults to out1 for the first Planned Output, out2 for the second, and so on.

3. Click **Apply**. The system saves the changes and returns you to the Program Instance screen.

- Planned Output Classification

Planned Output Classification

You must define default classification values for the Planned Output. The system applies these classifications to the actual output when it is generated by the execution of the Program. The classifications you define for a Planned Output determine who can see the actual output when it is generated, and where it appears in the navigation tree in the Reports tab in Oracle LSH.

For further information, see Classifying Outputs.

Defining PL/SQL Programs

See also Setting Up Oracle SQL Developer or SQL*Plus as an IDE

> **Note:**
>
> Each PL/SQL Source Code definition within a particular Program must have a unique Oracle package name.

This section includes information on:

- Writing Primary Source Code in PL/SQL
  When a Program is executed, the system launches its primary source code file.
- Testing PL/SQL Source Code
  To test PL/SQL code, you must first map the Table Descriptors to the Table instances and check in and install the Program instance and all the Table instances it reads from and writes to.
- Creating a PL/SQL Package Storage Program
  You can create PL/SQL Programs especially for the purpose of containing PL/SQL packages as reusable (sharable) Source Code definitions, so that Definers can more easily find them.
- Using a Sharable PL/SQL Package
  In Oracle Life Sciences Data Hub Programs of type PL/SQL or Oracle Report, you can create an instance of a Source Code definition stored in another Program and marked as Sharable.
- Compiling and Executing a PL/SQL Program
  The system compiles PL/SQL source code when you install the Program instance.
- Manipulating Documents through a PL/SQL Program
  You can write a PL/SQL Program to retrieve BLOBs (binary large objects such as documents created using the Microsoft Office Suite) from the database and use them, for example, as a Planned Output of the Program for inclusion in a Report Set.

Writing Primary Source Code in PL/SQL

When a Program is executed, the system launches its primary source code file.

You must use a specific syntax at the beginning of the PL/SQL source code and also write the source code in such a way that it calls every secondary Source Code instance you define and refers to all defined subcomponents by their Oracle name.

Required Syntax: Must Match Definitions: In the primary Source Code of a PL/SQL Program, the source code must begin by providing the Oracle Package name and Oracle Procedure name defined for the Source Code, and declare all Parameters defined in the Program with their data type, as shown in the following example, where the first Parameter is a number and the second Parameter is a varchar2:

```
create or replace package PACKAGE_NAME asprocedure PROCEDURE_NAME (parameter_1
number,parameter_2     varchar2);end PACKAGE_NAME;/create or replace package body
PACKAGE_NAME asprocedure MAIN (parameter_1 number,        parameter_2     varchar2)
isbegin
```

Required Security Syntax: There is a potential security hole in PL/SQL Programs because they can be executed directly in the database outside of Oracle Life Sciences Data Hub security.

Oracle Life Sciences Hub can prevent this if you add a specific code template. You should add this to the beginning of the initialization block of your primary PL/SQL source code, as shown below. When you first install the Program, the system compiles the PL/SQL source code and inserts the actual Program ID generated by the system for the Program. At runtime, the system checks that a database account corresponding to the Program ID has been created. The service instance creates this database account to allow execution of the PL/SQL packages. If the account exists, then the job has been created through proper channels and is allowed to proceed. If it has not, the system does not allow execution to proceed.

If you do not include the recommended code template, when you install the Program, the system looks for either `END;` or `END` *`package_name;`* beginning at the end of the source code, and inserts the security code at that point. However, at runtime the Program is allowed to run up until that point. Any statements that appear in the initialization block before the security code are allowed to execute.

Add the following template exactly as appears:

```
BEGIN  /*Package initialization here*/
/* LSH GENERATES SECURITY CODE HERE, DO NOT REMOVE THIS COMMENT. */
/* Define your package initialization here */
NULL;
```

Insert the above template into the package initialization block of the package body, as follows:

```
CREATE OR REPLACE PACKAGE pkg1 AS
/* define your procedures here */
PROCEDURE proc1;
END pkg1;
/
CREATE OR REPLACE PACKAGE BODY pkg1 AS
/* define your parameters here */

/* define your procedures here */
PROCEDURE proc1 IS
BEGIN
/* Define code here */
NULL;
END proc1;

BEGIN  /*Package initialization here*/
/* LSH GENERATES SECURITY CODE HERE, DO NOT REMOVE THIS COMMENT. */
/* Define your package initialization here */
NULL;
END pkg1;
```

The first time you install the Program, the system updates your source code by inserting the following code, including the actual program_id generated for the program by the system:

```
IF NVL(SYS_CONTEXT('CDR_RUNTIME', <program-id>), 'X') <> 'Y'
THEN
   RAISE_APPLICATION_ERROR(-20005, 'EXECUTE NOT enabled.');
END IF;
```

Subcomponent References in PL/SQL: You must refer to the defined subcomponents of the Program in your PL/SQL source code as follows:

- **Table Descriptors**. For each table you read from or write to in your source code, you must define a source or target Table Descriptor. Refer to each Table Descriptor as if it were a real database table, using its Oracle name. If the Table instance to which a Table Descriptor is mapped has a different name from the Table Descriptor, use the Table Descriptor's name, not the Table instance's.

  > **Note:**
  >
  > In a PL/SQL Program a Source Code and a Table Descriptor cannot have the same Oracle name.

- **Secondary Source Code**. Refer to secondary Source Code instances by their Oracle name.
- **Parameters**. You must create a defined Parameter for each input and output Parameter you use in your primary source code, and declare them in your source code. Refer to defined Parameters by their Oracle name.
- **Planned Outputs**. You must create a defined Planned Output for every output generated by the primary source code at execution, including the log file. Refer to each defined Planned Output by its File Name.

API for Ending PL/SQL Programs with a Status of Success, Warning, or Failure: Normally PL/SQL programs end with a status of Success unless there is a system failure or unhandled SQL exception. However, if you are using a Program in a Workflow, you may need to write your code so that the Program completes with a status of Warning or Failure, depending on circumstances. In a Workflow, you can use the completion status of a Program to determine which branch of activities to execute.

Oracle Life Sciences Hub ships with an API for this purpose called:

CDR_EXE_USER_UTILS.setCompletionStatus()

To call the package, enter one of the following lines of code in your source code exactly as it appears below, at the point where you want the Program to return a status of Success, Warning, or Failure:

```
CDR_EXE_USER_UTILS.setCompletionStatus(1);
CDR_EXE_USER_UTILS.setCompletionStatus(2);
CDR_EXE_USER_UTILS.setCompletionStatus(3);
```

CDR_EXE_USER_UTILS.setCompletionStatus(1) returns a status of Success.
CDR_EXE_USER_UTILS.setCompletionStatus(2) returns a status of Warning.
CDR_EXE_USER_UTILS.setCompletionStatus(3) returns a status of Failure.

Testing PL/SQL Source Code

To test PL/SQL code, you must first map the Table Descriptors to the Table instances and check in and install the Program instance and all the Table instances it reads from and writes to.

You can then execute the Program. If the Program writes data to tables, you can check the data; see Viewing Data.

If the Program generates a report, you can see the report in the Reports tab. You can link to the report and the log file from the Job ID link in the Job Executions section of your My Home tab.

Creating a PL/SQL Package Storage Program

You can create PL/SQL Programs especially for the purpose of containing PL/SQL packages as reusable (sharable) Source Code definitions, so that Definers can more easily find them.

You can group logically related packages in the same container Program.

You can then use these sharable PL/SQL packages in Oracle LSH Programs of type PL/SQL or Oracle Report (see Using a Sharable PL/SQL Package).

To create a storage Program for PL/SQL packages:

1. Create a Program of type PL/SQL. Give it a name and description that describe its purpose, such as "Sharable Demography Subroutines."
2. (Optional) Add a primary Source Code definition of type PL/SQL to test the packages. Set its **Sharable** flag to **No**.
3. Add the secondary Source Code definitions you want to share. You can create them and write the code from within the storage Program, or you can copy them from other Programs and paste them into the storage Program. They must all be of type PL/SQL.
4. Check in, install, and test the Program.
5. When you have tested each one, set its **Sharable** flag to **Yes**. You can change this setting without checking out the Program definition.

Using a Sharable PL/SQL Package

In Oracle Life Sciences Data Hub Programs of type PL/SQL or Oracle Report, you can create an instance of a Source Code definition stored in another Program and marked as Sharable.

Your company may have Programs created especially for the purpose of storing sharable PL/SQL packages.

To use a Sharable Source Code definition:

1. In a PL/SQL or Oracle Reports Program, create a secondary Source Code object as an instance of an existing Source Code definition (see Creating an Instance of an Existing Source Code Definition).
2. When you search for the definition source, specify the PL/SQL package storage Program as the Program search criterion.
3. You can select one or more of the PL/SQL packages. The system includes the source code for each package you select in the Source Code instance of your Program.

See Upgrading Source Code And Undoing Source Code Upgrades for information on how to upgrade Source Code instances pointing to a sharable Source Code definition.

Compiling and Executing a PL/SQL Program

The system compiles PL/SQL source code when you install the Program instance.

Therefore, when a Program containing PL/SQL packages is submitted for execution, the system does not need to compile its source code.

Manipulating Documents through a PL/SQL Program

You can write a PL/SQL Program to retrieve BLOBs (binary large objects such as documents created using the Microsoft Office Suite) from the database and use them, for example, as a Planned Output of the Program for inclusion in a Report Set.

You can also stream a BLOB into a Program.

Defining View Programs

This section includes information on:

- Creating Source Code for a View Program
- About View Table Instances
- About Table Descriptors in View Programs

The View-type Program creates a database view of data in multiple Table instances. When you install the View Program, the system creates a Table definition, a Target Table Descriptor and a Table instance object of type View; and maps the Table instance to the target Table Descriptor.

The resulting View Table instance created in the Work Area schema functions like any database view and always displays current data available in the source Table instances. You can combine data from various Table instances without storing the data multiple times in the database. Other Programs, Business Areas and Data Marts can also read from the View Table instance. View-type Programs cannot be executed and therefore you cannot create them under a Report Set or Workflow.

The status of a View Program is 'Installable' if the View Program contains both a Source Code and a source Table Descriptor. The View Program's Launch IDE behavior is the same as that of a PL/SQL Program.

- Creating Source Code for a View Program
  When you launch the View Program, the system launches its primary source code file.
- About View Table Instances
  You can only create a View Table instance by installing a View Program. You cannot create or update a View Table instance from the Oracle Life Sciences Data Hub User Interface or through APIs. Once created, the target Table Descriptor cannot be unmapped from its Table instance.
- About Table Descriptors in View Programs
  You cannot map source Table Descriptors of a View Program to a View Table instance.

Creating Source Code for a View Program

When you launch the View Program, the system launches its primary source code file.

The following is a format of a SELECT statement that can be customized and saved as Source Code in a View Program:

```
SELECT [ hint ] [ { { DISTINCT | UNIQUE } | ALL } ] select_list FROM table_reference
[, table_reference ]... [ where_clause ] [ hierarchical_query_clause ]
[ group_by_clause ] [ HAVING condition ] [ model_clause ] [ { UNION [ ALL ] |
INTERSECT | MINUS } (subquery) ] [ order_by_clause ]
```

The SELECT query in the View Program Source Code can only reference the source Table descriptors. It may call functions which are part of PL/SQL programs that are statically referencedby the View Programs.

A View Program cannot contain any definition, runtime parameter or Planned Outputs.

During installation, the system validates the SQL statement for any external references. Therefore, the SQL statement cannot reference any public object in the database.

About View Table Instances

You can only create a View Table instance by installing a View Program. You cannot create or update a View Table instance from the Oracle Life Sciences Data Hub User Interface or through APIs. Once created, the target Table Descriptor cannot be unmapped from its Table instance.

If you remove a View Program instance, its corresponding target Table instance is also removed.

If you modify the SELECT statement before installing the program, the system synchronizes the Target Table Descriptor and the Table instance at the time of install.

A View Table instance has full currency traceability and can be used as a source table for other Oracle Life Sciences Hub Programs, Data Marts and Business Areas. Every time you access the view's current data, a currency timestamp is applied to the View Table.

You can use the **Browse Data** UI to view data in the View Table instance. You can use the **Apply Snapshot Label** UI to select snapshot labels for all source Tables and apply a snapshot label to the target View Table instance. You can remove a snapshot label using the **Manage Snapshot Label** UI.

The validation status of a View Table instance must always be equal to or lower than that of the source Table instances.

Any destructive change to a source Table Instance invalidates the View Table instance. You must update and reinstall the View Program to resolve differences. The system does not allow you to remove source Tables if the validation status of a View Table instance is Production.

- Blinding Status
  The blinding status of the View Table instance is determined by that of the Table Instances mapped to the source Table Descriptors in the View Program.

Blinding Status

The blinding status of the View Table instance is determined by that of the Table Instances mapped to the source Table Descriptors in the View Program.

If you have blind-break privileges on source Table instances that support blinding, you can authorize a view based on those Tables. You can authorize reading from blinded sources only if all the source Table Descriptors are mapped. Use the **Authorize Read**

from Blinded Sources radio button on the View Program instance Properties page to authorize a View Table. By default, the setting is **No**.

If mapping of any source Table Descriptors becomes incomplete or new Descriptors are added, the **Authorize Read from Blinded Sources** setting is automatically set to **No**. When the authorize setting is modified, the Program instance is implicitly checked out.

About Table Descriptors in View Programs

You cannot map source Table Descriptors of a View Program to a View Table instance.

In a View Progam, if the target Table Descriptor and its corresponding View Table instance exist, and if the SELECT query in the Source Code is modified, installation automatically updates the corresponding Table definition in the Application Area containing the Work Area. The system then updates the Table Descriptor and Table instance to refer to the modified Table definition.

If the Table definition for the Table Descriptor is updated during installation, the system checks if the Table definition exists in the local Application Area. If not, a copy of the Table definition is created in the local Application Area and modified.

Defining SAS Programs

See also: Setting Up SAS as an IDE

This section includes information on:

- SAS Program Development Process
  There are three basic ways to use SAS source code in Oracle Life Sciences Data Hub.
- Connecting to SAS
  There are three ways to connect to SAS for Program development: Connected Mode, SAS Connected Mode with Work Area Data, and Disconnected Mode. You can specify the mode you want to use in your User Preferences, although your choices may be restricted by your company.
- SAS Program and Source Code Types
  Oracle Life Sciences Data Hub supports three types of SAS Programs and handles each one differently during execution:
- Writing SAS Primary Source Code
  Create a Source Code definition of type Program in a SAS Program to hold the source code that accomplishes the business purpose of the Program: merging or transforming data and/or producing one or more figures, listings, or table reports.
- Creating a SAS Macro Catalog
  Oracle Life Sciences Data Hub includes the Program type SAS macro Catalog especially for the purpose of storing SAS macros that are approved for reuse.
- About SAS Format Catalogs in the Oracle Life Sciences Data Hub
  A SAS format translates short data value codes like zero (0) and one (1) to meaningful data values such as Male and Female or Yes and No.
- Creating a SAS Format Catalog
  To create a SAS Format Catalog in Oracle Life Sciences Data Hub:

- Calling an API to Capture Output Parameter Values

SAS Program Development Process

There are three basic ways to use SAS source code in Oracle Life Sciences Data Hub.

- Open SAS as an IDE from Oracle Life Sciences Data Hub
  If you have the SAS client installed on your PC, you can launch the SAS integrated development environment (IDE) from an Oracle Life Sciences Hub Program instance.
- Upload Existing SAS Programs to Oracle Life Sciences Data Hub
  You may have many legacy SAS programs that you want to use on Oracle Life Sciences Hub data.
- Enter Source Code Directly in the Oracle Life Sciences Data Hub Source Code Definition
  When you create a new Source Code definition and instance at the same time, you can type or copy and paste source code text directly into the large Source Code field.

Open SAS as an IDE from Oracle Life Sciences Data Hub

If you have the SAS client installed on your PC, you can launch the SAS integrated development environment (IDE) from an Oracle Life Sciences Hub Program instance.

If you plan to use the SAS (IDE) to develop an Oracle Life Sciences Hub Program, before you launch the IDE:

- Define and map the source Table Descriptors you need; see Defining and Mapping Table Descriptors.
- Install the Program instance; see Installing a Work Area and Its Objects.

Oracle Life Sciences Hub then downloads the data views or files to SAS (depending on the type of connection you are using; see Connecting to SAS) when you launch SAS and you can read the data as necessary while you write the source code in SAS.

You can go back and forth between working in SAS and working in Oracle Life Sciences Hub as you develop a Program. For example, if you declare an input or output parameter in your SAS code, you can immediately go to Oracle Life Sciences Hub and create the required corresponding Parameter in the Oracle Life Sciences Hub Program, and then go back to writing SAS code.

When you are ready, go to the Source Code definition in the Oracle Life Sciences Hub Program instance and upload your SAS source code.

Upload Existing SAS Programs to Oracle Life Sciences Data Hub

You may have many legacy SAS programs that you want to use on Oracle Life Sciences Hub data.

You can upload an existing SAS program to a Source Code definition in an Oracle Life Sciences Hub SAS Program and create defined Parameters, Source Codes, Table Descriptors, and Planned Outputs as required by Oracle Life Sciences Hub for the SAS source code.

Enter Source Code Directly in the Oracle Life Sciences Data Hub Source Code Definition

When you create a new Source Code definition and instance at the same time, you can type or copy and paste source code text directly into the large Source Code field.

Connecting to SAS

There are three ways to connect to SAS for Program development: Connected Mode, SAS Connected Mode with Work Area Data, and Disconnected Mode. You can specify the mode you want to use in your User Preferences, although your choices may be restricted by your company.

- **Connected Mode**. Your PC has the SAS client and SAS Access to Oracle installed and is connected to the Oracle Life Sciences Data Hub database through a network. When you launch SAS from a Program instance, Oracle Life Sciences Hub downloads views based on the Table Descriptors defined in the Program. You write your program locally on the SAS client, using the views to see data in Oracle Life Sciences Hub. You cannot write data to Oracle Life Sciences Hub Table instances. If you run the Program locally, you write data to local data set files. When you are ready, go to the Program instance in Oracle Life Sciences Hub, upload the SAS source code and upload any target SAS data sets you have created as Table Descriptors.
- **SAS Connected Mode with Work Area Data**. This mode is the same as Connected mode except that it connects to the Work Area schema in the database. From SAS, you can browse views of current data in all Table instances in the Work Area, not just the Table instances linked to Table Descriptors of the Program.
  You must use the SAS Access to Oracle tool to connect to Oracle Life Sciences Hub.
- **Disconnected Mode**. Your PC has the SAS client installed and is connected to the Oracle LSH database through a network. When you launch SAS from a Program instance, Oracle LSH downloads data sets with the same structure as the Program's Table Descriptors. In addition, Oracle Life Sciences Hub downloads the actual data contained in the Table instances to which the Table Descriptors are mapped. You can write your program, working locally on the downloaded data. Oracle Life Sciences Hub creates a directory structure on your personal computer based on the location of the Program, starting with the Domain (if you are using multiple levels of Domains, all are represented): *Drive:*/CdrWork/*your_LSH_database_acount_name/Domain_name_(all_existing_domains)/Application_Area_name/Work_Area_name/Program_instance_name/Program_instance_version/Table_Descriptor_SAS_libname/*data set file.

  For Source Codes, Oracle Life Sciences Hub creates directories on your PC to contain the source code files. The system creates one directory for Source Code definitions of type Program and another for those of type Macro:

  – *Drive:*/CdrWork/*your_LSH_database_acount_name/Domain_name_(all_existing_domains)/Application_Area_name/Program_definition_name/Program_version*Programs/source code files.

– *Drive:/*CdrWork*/your_LSH_database_acount_name/Domain_name_(all_existing_domains)/Application_Area_name/Program_definition_name/Program_version/*Macros/source code files.

When you are ready, go to the Program instance in Oracle Life Sciences Hub, upload the SAS source code and upload any target SAS data sets you have created as Table Descriptors.

> **Note:**
>
> It is possible to work on the same Program in different modes at different times. However, if you work first in Disconnected mode, so that the system downloads data to your personal computer, and then change to Connected mode, you may get an error that the source data set already exists. In this case, the system continues to point to the local data set instead of live data in Oracle Life Sciences Hub.
>
> To avoid this problem, delete or move the data sets on your personal computer that were downloaded from Oracle Life Sciences Hub.

> **Note:**
>
> Developing SAS code is an option only for customers who purchase SAS separately from Oracle Life Sciences Hub. See Setting Up Integrated Development Environments (IDEs) for instructions on how to set up SAS to work with Oracle Life Sciences Hub.

SAS Program and Source Code Types

Oracle Life Sciences Data Hub supports three types of SAS Programs and handles each one differently during execution:

- SAS Program
- SAS Macro Catalog
- SAS Format Catalog

SAS Macro Catalogs and SAS Format Catalogs can be referenced by primary or secondary Source Code instances in a SAS Program and are compiled each time the Program is executed, before the primary Source Code is launched.

- SAS Program
  Define an Oracle Life Sciences Data Hub Program of type SAS Program to hold the source code of a normal SAS program that manipulates data or generates one or more reports.
- SAS Macro Catalog
  An Oracle Life Sciences Data Hub Program of type SAS Macro Catalog is intended to store a set of macros that are approved for reuse in a variety of SAS Programs.

- SAS Format Catalog
  An Oracle Life Sciences Data Hub Program of type SAS Format Catalog is intended to store a set of formats that are approved for reuse in a variety of SAS Programs.

SAS Program

Define an Oracle Life Sciences Data Hub Program of type SAS Program to hold the source code of a normal SAS program that manipulates data or generates one or more reports.

Upload this SAS source code to Oracle Life Sciences Hub as primary source code. In this primary source code you can call SAS macros or formats stored in Oracle Life Sciences Hub Programs of type SAS Macro Catalog or SAS Format Catalog, or stored in the same Program as secondary Source Code of type .

Before you launch the SAS development environment to write source code, you must define a Program's source Table Descriptors and map them to Table instances so that Oracle Life Sciences Hub can download the views or data for you to use.

In a SAS Program you can have two types of source code:

- **Program**. Source Code of type Program is intended to hold the source code that accomplishes the business purpose of the Program: merging or transforming data and/or producing one or more figures, listings, or table reports. You must designate the Source Code that serves this purpose as the primary Source Code so that the system sends it to the SAS engine for execution. The actual SAS source code file contained in the Source Code definition can call other Source Codes of type Program or Macro, or Oracle Life Sciences Hub Programs of type SAS Macro Catalog or SAS Format Catalog.
- **Macro**. You can define a macro specifically for use within a particular Program. These Source Code definitions can be displayed in any order. The system compiles them before each execution and executes them in the order they are called by the primary source code.

See Writing SAS Primary Source Code.

SAS Macro Catalog

An Oracle Life Sciences Data Hub Program of type SAS Macro Catalog is intended to store a set of macros that are approved for reuse in a variety of SAS Programs.

You can group a set of macros with related functions in a single Catalog; for example, demography macros. In a SAS Macro Catalog Program you can have two types of source code:

- **Macro**. A Source Code definition of type Macro to hold the source code for a single SAS macro. Set each macro's **Sharable** flag to **Yes**.
- **Program**. One or more Source Code definitions of type Program to test the macros. Set its **Sharable** flag to **No**. This Source Code must be listed in the first (primary) position so that Oracle Life Sciences Hub sends it to the SAS engine to test the macros.

SAS Format Catalog

An Oracle Life Sciences Data Hub Program of type SAS Format Catalog is intended to store a set of formats that are approved for reuse in a variety of SAS Programs.

You can group a set of formats with related functions in a single Catalog; for example, demography formats. In a SAS Format Catalog Program you can have two types of source code:

For Source Code of type Macro in any Program type, you must upload the source code, not the compiled binary file. The system compiles the macros defined in a Program before each execution of a Program's primary source code.

- **Macro**. A Source Code definition of type Macro to hold additional source code to support the format building steps. Set each macro's **Sharable** flag to **Yes**.
- **Program**. One or more Source Code definitions of type Program to test the formats. Set its **Sharable** flag to **No**. This Source Code must be listed in the first (primary) position so that Oracle Life Sciences Hub sends it to the SAS engine to test the formats.

Writing SAS Primary Source Code

Create a Source Code definition of type Program in a SAS Program to hold the source code that accomplishes the business purpose of the Program: merging or transforming data and/or producing one or more figures, listings, or table reports.

You must designate the Source Code that serves this purpose as the primary Source Code so that Oracle Life Sciences Data Hub sends it to the SAS engine for execution. Its source code can call other Source Codes of type Program or Macro contained in the same SAS Program, or Oracle Life Sciences Hub Programs of type SAS Macro Catalog or SAS Format Catalog.

If you plan to launch the SAS development environment from Oracle Life Sciences Hub to write source code, you must first define a Program's source Table Descriptors, map them to Table instances, and install the Program and Table instances so that Oracle Life Sciences Hub can download the views or data for you to use.

> **Note:**
>
> Do not include the string `error:` in any SAS source code. Oracle Life Sciences Hub searches the Program execution log file for the string "Error:" and errors out the Program execution if it finds the string. The source code of the Program is copied into the log file. Therefore if you include "Error:" in your source code, the Program will fail.

Subcomponent References in SAS: You must refer to the defined subcomponents of the Program in your SAS source code as described in the following sections:

Table Descriptors: Oracle Life Sciences Hub Tables and Table Descriptors are compatible with SAS data sets. The Table is equivalent to a data set, and Table Columns are equivalent to a data set's variables.

Syntax. Write to each Table Descriptor defined within the Program as if it were a data set, using the syntax *SAS_library_name.SAS_name*. You must read from and write to the Table Descriptor, not the Table instance; if the name of the Table Descriptor or its Columns differ from the Table instance's, use the Table Descriptor's.

Target As Dataset. Because Program source code must write to Table Descriptors, and Table Descriptors are views, you should use Proc SQL statements to write to tables in Oracle Life Sciences Hub. However, Oracle Life Sciences Hub provides a feature to allow you to use existing SAS Programs written with data statements. The Target As Dataset attribute is available only in SAS Programs, and only for target Table Descriptors.

If you set this attribute to **Yes**, Oracle Life Sciences Hub adds a processing step to enable SAS data statements to write to Oracle Life Sciences Hub Table instances. This extra processing step results in slower performance but allows you to use existing programs.

> **Tip:**
>
> You can improve processing performance by selecting Target as Dataset and using the SQL*Loader. See SQL*Loader Processing for SAS Programs.

Select **No** if the Program's source code uses Proc SQL statements to write to tables. This results in optimal performance.

SAS Secondary Source Code Instances: In an Oracle Life Sciences Hub SAS Program you can create secondary Source Code instances of four types:

- **SAS Macro Catalog**. To use any of the macros included in Oracle Life Sciences Hub SAS Macro Catalog in a Source Code instance, create an instance of the catalog Source Code in your Program. Immediately before each execution of the Oracle Life Sciences Hub SAS Program the macros are compiled in the SAS work library. You can call them by name from the primary Source Code.
- **SAS Format Catalog**. To use any of the formats included in an Oracle Life Sciences Hub SAS Format Catalog in a Source Code instance, create a Source Code instance of the whole catalog in your Program. Immediately before each execution of the Oracle Life Sciences Hub SAS Program the formats are compiled in the SAS work library. You can call them by name from the primary Source Code.
- **Macro**. You can create a macro especially for use in the same Program where your primary Source Code is located. You can also create an instance of a sharable Source Code definition of type Macro from another SAS Program. Refer to individual Source Codes of type Macro in your primary source code by their name.
- **Program**. You can use another Oracle Life Sciences Hub SAS Source Code of type Program (not an Oracle Life Sciences Hub Program of type SAS Program) as an Include. If the Source Code definition is located in the same Program as your primary Source Code, refer to it by its name. If the Source Code definition is located in a different Program, refer to it by its SAS File Reference Name.

Paramters: For every input or output parameter in your SAS primary source code, you must define a Parameter in Oracle Life Sciences Hub and refer to it by its name in your SAS code. See Defining Parameters .

Planned Outputs: You must define a Planned Output to hold each report to be generated by a Program and refer to each one in the source code by its SAS File Reference Name. Oracle Life Sciences Hub automatically generates a Planned Output for the log file when you create the first Source Code in the Program. See Defining Planned Outputs.

You can successfully execute a SAS Program through Oracle Life Sciences Hub even if there are unplanned Outputs. If the SAS Program's validation status is Development, the system automatically generates a Planned Output with a file reference name created from the first eight characters of the Output file name in the source code. Ensure that the Output file name in the Source Code is enclosed in double quotations for Oracle Life Sciences Hub to identify.

If the required Output is a SAS data set, Oracle Life Sciences Hub can generate a Planned Output if the SAS Program's validation status is Development. You must define the new SAS data set inside the library named "Target". Oracle Life Sciences Hub then treats the Output data set as the SAS Program's target Table. The system compares the SAS data set filenames with those of the target Table Descriptors and if they match, loads data from the SAS data set into the mapped Table instance.

If the system does not find a target Table Descriptor, it checks out the Program, adds a new target Table Descriptor to it, creates a new Table instance in the Work Area and maps it to the newly created target Table Descriptor before installing the Program and the Table instance. Data is then loaded from the SAS data set into the mapped Table instance. Oracle Life Sciences Hub triggers separate jobs for each of these processes. The system does not support Proc SQL and Insert commands in the generation of unplanned data set Outputs.

Subsequent changes in source data set Table Descriptors must be matched in the target Table manually.

- Using a SAS Macro Catalog
  To use any of the macros contained in an Oracle Life Sciences Data Hub SAS Macro Catalog, you create a Source Code instance in the Program from which you need to call them.
- Using a SAS Format Catalog

Using a SAS Macro Catalog

To use any of the macros contained in an Oracle Life Sciences Data Hub SAS Macro Catalog, you create a Source Code instance in the Program from which you need to call them.

Do the following:

1. In the Oracle Life Sciences Hub SAS Program where you need to use one or more of the macros in the Catalog, create a Source Code as an instance of an existing definition.
2. In the Search screen, choose the **Domain** or **Domain** and **Application Area** where the Macro Catalog you need is located, and select the **SAS Macro Catalog** radio button. If you know the exact name of the Macro Catalog you need, enter it in the **Name** field.
3. Click **Go**. The system returns the Macro Catalog(s) that satisfy the search criteria —or, if you entered the exact name of a Macro Catalog, returns only that one.

4. Select a Macro Catalog: select its box in the **Select** column and click the **Select** button. The system adds an instance of the Catalog, including all the macros it contains, and returns you to the Source Code screen.

You can now use any of the macros in your Program. At execution they are added to your work library and you can call them by name from the primary source code.

Using a SAS Format Catalog

If your SAS code operates on data sets that require SAS formats for the proper expression of their data, you must include the formats in yourOracle Life Sciences Data Hub SAS Program (see Defining SAS Programs).

When you create a SAS Program in Oracle Life Sciences Hub that needs to use an Oracle Life Sciences Hub SAS Format Catalog, do the following:

1. In the Oracle Life Sciences Hub SAS Program, create a secondary Source Code object as an instance of an existing Source Code definition (see Creating an Instance of an Existing Source Code Definition).

   If the Format Catalog includes a Table Descriptor for use as a static reference, select the **Static Reference** radio button.

2. In the Search screen, choose the **Domain** or **Domain** and **Application Area** where the Format Catalog is located and select the **SAS Format Catalog** radio button. If you know the exact name of the Catalog you need, you can enter it.
3. Press **Go**. The system displays all the SAS Format Catalogs in the location you specified—or, if you supplied the exact name of a Catalog, lists only that Catalog.
4. Select the **Catalog** you want by selecting the box next to it in the **Select** column and click the **Select** button. The system adds the Catalog as a Source Code instance to your Program and returns you to the Program's Properties screen.

> **Note:**
>
> If you create an instance of a static reference Source Code definition, Oracle Life Sciences Hub adds the Table Descriptor(s) to the Program.

You can now use any of the formats in your Program. At execution they are added to your work library and you can call them by name from the primary source code, for example:.

Creating a SAS Macro Catalog

Oracle Life Sciences Data Hub includes the Program type SAS macro Catalog especially for the purpose of storing SAS macros that are approved for reuse.

You can group logically related macros in each SAS Macro Catalog.

You must upload the source code file, not the compiled binary file. When the Program is submitted for execution, the system compiles its macros before executing the primary source code.

- Creating a SAS Macro Catalog
  To create a SAS Macro Catalog in Oracle Life Sciences Data Hub:
- Nesting SAS Macros

Creating a SAS Macro Catalog

To create a SAS Macro Catalog in Oracle Life Sciences Data Hub:

1. Create a Program of type SAS Macro Catalog. Give it a name and description that describe its purpose (see Creating Source Code).

   > **Note:**
   >
   > You cannot set a Static-reference Source Code as primary. Also, you cannot set a Source Code that points directly to a Program definition as primary.

2. For each SAS macro you want to store in the Catalog, create a Source Code definition of type Macro.
3. If the macro does not already exist, create it either in SAS or in the Source Code box. If the source code is on a local computer, upload the source code file (not the compiled binary file) from SAS to Oracle LSH.
4. Set each macro's **Sharable** flag to **Yes**.

   > **Note:**
   >
   > You cannot set any macro Source Code instance that points directly to a Program definition as Sharable.

5. (Optional) Add a primary Source Code definition of type SAS Program to test to SAS macros you store in the Catalog. Set its **Sharable** flag to **No**. Write the source code and upload if necessary.

   > **Note:**
   >
   > This source code is not compiled when the Program is executed because its Sharable flag is set to No.

6. Test the macros and promote the Catalog to a higher validation status according to your company's policies.

Nesting SAS Macros

It is possible to create macros that reference other macros contained in a different SAS Macro Catalog in Oracle Life Sciences Data Hub. In this case, you include as a Source Code instance in your Oracle Life Sciences Hub SAS Program only the SAS Macro Catalog that contains the macro your primary source code references. When the Program is submitted for execution, the system compiles the macro specified in the Program and automatically finds and compiles the macro referenced by the Program's macro. You cannot use more than two layers of macros; a macro referenced by another macro cannot reference yet other macro.

About SAS Format Catalogs in the Oracle Life Sciences Data Hub

A SAS format translates short data value codes like zero (0) and one (1) to meaningful data values such as Male and Female or Yes and No.

Using SAS formats, you can store a minimum amount of data and call the format to correctly display the data in a report. The format itself can consist of source code containing hardcoded values, such as:

Alternatively, the format can consist of a simple call to an Oracle Life Sciences Hub Table instance that contains the relationships between the short stored values and the meaningful display values.

In this case the format refers to a data set, now converted to an Oracle Life Sciences Hub Table called Standard Formats that contains data such as:

Table 5-1 Format Table Example

Format Name	Stored Code	Display Value
$Sex.	0	Male
$Sex.	1	Female
$YesNo.	0	No
$YesNo.	1	Yes

You can create the Oracle Life Sciences Hub Table in several ways, including:

- uploading a format data set from SAS using a Load Set
- modifying an Oracle Clinical Discrete Value Group (DVG) table (loaded into Oracle Life Sciences Hub by the Oracle Clinical Global Library adapter)
- uploading a table from any integrated external system and modifying as necessary

A SAS format data set contains all the information required to define a format, such as the format name; its starting and ending value; minimum, maximum, and default length; and so on as columns or variables.

Creating a SAS Format Catalog

To create a SAS Format Catalog in Oracle Life Sciences Data Hub:

1. Create an Oracle LSH Program of type SAS Format Catalog. Give it a name and description that describe its purpose.

> **Note:**
>
> You cannot set a Static Reference Source Code as primary. Also, you cannot set a Source Code that points directly to a Program definition as primary.

2. If any of the formats in the Catalog are table-dependent, add the necessary Table Descriptor(s) using the Oracle Life Sciences Hub Table definition that you created for formats as the source Table definition (see Creating a Table Descriptor).
3. Map any Table Descriptors to Table instances (see Mapping Table Descriptors to Table Instances).
4. Add a Source Code definition of type Program that contains the format source code (see Creating Source Code).

   If the format is Table-dependent, the source code must use the Table Descriptor as input and include a SAS Proc format to create the format catalog in SAS.

   You can also add Source Codes of type Macro to support the format building steps if necessary.
5. Set the Source Code's **Sharable** flag to **Yes**.

> **Note:**
>
> You cannot set any Source Code instance that points directly to a Program definition as Sharable.

6. Repeat as necessary; you can have multiple format Source Codes in a single Oracle Life Sciences Hub SAS Format Catalog.
7. Apply. The system saves the Source Code definition(s) and instances in the database and returns you to the Program's main page.
8. Install the Program in the database (see Running a Work Area Installation).

Calling an API to Capture Output Parameter Values

You can call a public Oracle Life Sciences Data Hub API from SAS source code to capture the values of output Parameters in a SAS Program contained in a Report Set or Workflow for the purpose of passing their value during execution to another Program in the same Report Set or Workflow (see Setting Up Parameter Value Propagation).

If you set up value propagation in a Report Set or Workflow, you must call an API from each SAS Program whose output Parameter values you need to capture. You must call the API once for each Parameter value you need. You may want to add the API call to every SAS Program that contains output Parameters in case you later add the Program to a Report Set or Workflow and want to use the output Parameter value in value propagation. The API package procedure name is: Cdr_Pub_Exe_User_Utils.setOutputParams.

The example below uses a PL/SQL wrapper to call the API. In this way you can call the API multiple times and only connect to the database once from SAS, and only two arguments are required for each output Parameter that you want to send back to Oracle Life Sciences Hub:

- **pi_vparamName**. Enter the Name of the output or input/output Parameter whose value you want to capture.
- **pi_vparamValue**. This procedure parameter receives the value of the Program Parameter you specified as the value of pi_vparamName.

You can use the following code to call the API. Use %sysget (as shown) to get the required values rather than hardcoding the values in the code.

```
SAS code
--------------------------------------------------------------------------------
Proc SQL;
/*set the job context then send the output value*/
connect to oracle (user=%sysget(CDR_SCHEMA) pass=%sysget(CDR_PASSWD)
path=%sysget(CDR_DB) );

/* pass output parameter back to LSH */
execute(exec my_plsql_package.setOutputParams(
'MyParamName'
,'My Param Value'
)
by oracle ;

PL/SQL code
--------------------------------------------------------------------------------
CREATE OR REPLACE PACKAGE my_plsql_package AS
Procedure setOutputParams(
pi_vParamName IN varchar2
,pi_vParamValue IN varchar2
);
END my_plsql_package;
/

CREATE OR REPLACE PACKAGE BODY my_plsql_package AS
Procedure setOutputParams(
pi_vParamName IN varchar2
,pi_vParamValue IN varchar2
) IS
return_status VARCHAR2(10);
msg_count NUMBER;
msg_data VARCHAR2(2000);
BEGIN
   Cdr_Pub_Exe_User_Utils.setOutputParams(p_api_version => 1
               ,p_init_msg_list => Cdr_Pub_Def_Constants.G_FALSE
               ,p_commit => Cdr_Pub_Def_Constants.G_FALSE
               ,p_validation_level => Cdr_Pub_Def_Constants.G_VALID_LEVEL_FULL
               ,x_return_status => return_status
               ,x_msg_count => msg_count
               ,x_msg_data => msg_data
               ,pi_vparamName => pi_vParamName
               ,pi_vparamValue => pi_vParamValue) ;
   IF return_status <> 'S' THEN
      RAISE_APPLICATION_ERROR(-20200,'Failed to call
Cdr_Pub_Exe_User_Utils.setOutputParams: '||msg_data);
   END IF ;
END setOutputParams;
END my_plsql_package;
/
```

Defining Oracle Reports Programs

You can use Oracle Reports as an integrated development environment to develop reports in Oracle Life Sciences Data Hub, launching Oracle Reports from a Program definition.

Oracle Life Sciences Hub Programs of type Oracle Reports can have two types of Source Code: Oracle Reports, which are uploaded from Oracle Reports, and PL/SQL, for subroutines; see Creating a PL/SQL Package Storage Program. (See also: Setting Up Integrated Development Environments (IDEs).

Oracle Reports Builder includes the following features:

- A query builder with a visual representation of the specification of SQL statements to obtain report data
- Wizards that guide you through the report design process
- Default report templates and layouts that can be customized to meet your organization's reporting needs
- The ability to generate code to customize how reports will run
- A Live Previewer that allows you to edit report layouts in WYSIWYG mode
- An integrated chart builder that helps you to graphically represent report data
- Web publishing tools that dynamically generate web pages based on your corporate data
- Other standard report output formats like HTML, PDF, Postscript, and ASCII (to make use of Oracle Life Sciences Hub's Publishing Light features, you must use PDF)

For information on using Oracle Reports Builder, see the Oracle Reports documentation:

1. Go to Oracle documentation at

   http://www.oracle.com/technology/documentation/index.html

2. Go to the URL for the *Oracle Reports Developer Reports Builder* manual for Oracle Reports 6i, which is the release included with the Oracle Life Sciences Hub technology stack:

   http://download-west.oracle.com/docs/pdf/A73172_01.pdf

> **Note:**
>
> This URL is correct as of the date of publication. If you have trouble with these instructions, try My Oracle Support.

Defining Informatica Programs

Using Oracle Life Sciences Data Hub Informatica Programs, you can access Informatica tools from within Oracle Life Sciences Hub to create and edit Informatica

mappings and workflows; and execute Informatica workflows from within Oracle Life Sciences Hub, on Oracle Life Sciences Hub data.

See also: Setting Up Informatica as an IDE

This section contains the following topics:

- Creating a New Informatica Program
  To define a new Informatica Program, do the following:
- Using Your Existing Informatica Mappings and Workflows
  If you want to use your existing Informatica mappings and workflows from the first time you install the Oracle Life Sciences Data Hub Informatica Program, do the following:
- Creating and Synchronizing Source Code
  An Oracle Life Sciences Data Hub Informatica Program's Source Code holds the Informatica mapping and workflow files.
- Using PL/SQL Source Code in an Oracle Life Sciences Data Hub Informatica Program
  Oracle Life Sciences Hub supports PL/SQL programs for Informatica through statically shared Oracle Life Sciences Hub PL/SQL Source Code.
- Updating Table Descriptors
  Oracle Life Sciences Data Hub Informatica Programs can read Oracle Life Sciences Hub Table instances but cannot write to target Table instances.
- Setting Informatica Program Parameters
  This section contains the following topics:
- Selective Index Management
  Use the Oracle Life Sciences Data Hub public API for selective index management to:
- Adding Planned Outputs
  No Planned Outputs are required for Informatica Programs.
- Informatica Integration
  This section contains information on the following:

Creating a New Informatica Program

To define a new Informatica Program, do the following:

1. After you create the Oracle Life Sciences Data Hub Informatica Program definition and instance, create Table Descriptors in it and map them. This readies the Program for installation. See Defining Table Descriptors for more information on Table Descriptors.
2. Install the Program and check it out.
3. Click **Launch IDE** to start Informatica's PowerCenter Designer. You can create your mappings in the PowerCenter Designer. You can access other Informatica components from the PowerCenter Designer; for example, you can go to the Informatica Workflow Manager from the PowerCenter Designer to create Informatica workflows.

> **Note:**
>
> When you launch Informatica PowerCenter Designer from an Oracle Life Sciences Hub Informatica Program for the first time, you have to configure the Oracle Life Sciences Hub Informatica Repository and connect to it. Consult your Informatica Administrator for more information.

See IDE Launch Settings for information on data access settings for IDEs. See the appropriate Informatica documentation for information on using Informatica.

4. Export the mappings and workflows from Informatica when done and upload the resultant XML files into the Source Code of the corresponding Oracle Life Sciences Hub Informatica Program. See Creating and Synchronizing Source Code.

See Informatica Integration for information about what happens behind the scenes when you check out, check in, and launch an Informatica Program in Oracle Life Sciences Hub.

Using Your Existing Informatica Mappings and Workflows

If you want to use your existing Informatica mappings and workflows from the first time you install the Oracle Life Sciences Data Hub Informatica Program, do the following:

1. From Informatica, export the mappings and workflows. Informatica generates an XML file for each mapping and each workflow.
2. In the Oracle Life Sciences Hub Informatica Program, create a Source Code definition and instance and upload the XML files into the Source Code definition. See Creating a New Source Code Definition and Instance for instructions.
3. Install the Oracle Life Sciences Hub Program. Oracle Life Sciences Hub creates a folder in Informatica (in the same format as described above) and imports the mapping and workflow files into this Informatica folder.

For more information on installing Programs in general, see Installing Program Instances.

Creating and Synchronizing Source Code

An Oracle Life Sciences Data Hub Informatica Program's Source Code holds the Informatica mapping and workflow files.

You must create a new Source Code definition and instance when you upload Informatica files for the first time. See Creating a New Source Code Definition and Instance for instructions.

When you make changes to mappings and workflows in Informatica, you must export the mappings and workflows from Informatica when done and upload the resultant XML files into the Oracle Life Sciences Hub Informatica Program's Source Code.

> **Note:**
>
> Informatica exports the mappings and workflows into separate XML files. When uploading the XMLs into the Oracle Life Sciences Hub Informatica Program's Source Code, upload the mapping XML before the workflow XML. This is because when you launch Informatica from the Oracle Life Sciences Hub Informatica Program, the files are imported into Informatica in the same order in which you uploaded them into the Source Code definition, and Informatica needs the mapping XML first.
> Oracle Life Sciences Hub Informatica Programs do not use the Primary and Secondary classification for the Source Code objects, unlike other Oracle Life Sciences Hub Programs.

You can also use PL/SQL Source Code in an Oracle Life Sciences Hub Informatica Program by creating a Source Code instance that refers to an installed Oracle Life Sciences Hub PL/SQL Source Code definition. See Using PL/SQL Source Code in an Oracle Life Sciences Data Hub Informatica Program.

> **Note:**
>
> Do not edit the XML files from within Oracle Life Sciences Hub.

Using PL/SQL Source Code in an Oracle Life Sciences Data Hub Informatica Program

Oracle Life Sciences Hub supports PL/SQL programs for Informatica through statically shared Oracle Life Sciences Hub PL/SQL Source Code.

See Creating and Using Static Reference Source Code.

To use PL/SQL Source Code in an Oracle Life Sciences Hub Informatica Program, do the following:

1. Create an Oracle Life Sciences Hub PL/SQL Program, create a Source Code definition and instance in this Program, upload or enter valid PL/SQL code in the Source Code definition, and install the Program. See Defining PL/SQL Programs.
2. In the Oracle Life Sciences Hub Informatica Program, create a Source Code object as an instance of an existing Source Code definition (see Creating an Instance of an Existing Source Code Definition.
3. When you search for the definition source, set **Static Reference** to **Yes**.

   > **Note:**
   >
   > You cannot add non-statically shared PL/SQL Source Code in Oracle Life Sciences Hub Informatica Programs.

4. Select the PL/SQL Source Code definition you want to use and click **Apply**.

If you create a Source Code instance of PL/SQL Source Code as a static reference (referring to the Source Code definition of an installed PL/SQL Program instance) in your Oracle Life Sciences Hub Informatica Program, and this PL/SQL Program needs to read the same source Tables as the Oracle Life Sciences Hub Informatica Program, copy the Oracle Life Sciences Hub Informatica Program and the PL/SQL Program that contains the statically shared Source Code, remove the original shared Source Code from the Oracle Life Sciences Hub Informatica Program and replace it with the Source Code from the copied PL/SQL Program, and map both the Informatica and the PL/SQL Program to new Table Instances.

Updating Table Descriptors

Oracle Life Sciences Data Hub Informatica Programs can read Oracle Life Sciences Hub Table instances but cannot write to target Table instances.

When you launch Informatica, Oracle Life Sciences Hub creates temporary Table instances to enable you to execute Informatica mappings from within Informatica (as opposed to from Oracle Life Sciences Hub). These temporary Table instances are only available for the current Informatica session. If you make any changes to the structure of these tables through Informatica, you must make the same changes manually in Oracle Life Sciences Hub.

See Defining Table Descriptors and Mapping Table Descriptors to Table Instances .

Oracle Life Sciences Hub Informatica Programs support indexes on Table instances and also allow selective index management in addition to recreating indexes for all the Tables. See Selective Index Management.

Setting Informatica Program Parameters

This section contains the following topics:

- User-Defined Parameters
  You must define a corresponding Parameter with the same name and type in the Oracle Life Sciences Data Hub Informatica Program for each parameter you use for mappings in Informatica.
- Predefined Parameters
  The Informatica adapter has the following predefined Parameters:

User-Defined Parameters

You must define a corresponding Parameter with the same name and type in the Oracle Life Sciences Data Hub Informatica Program for each parameter you use for mappings in Informatica.

You can pass values to Parameters when executing the Oracle Life Sciences Hub Informatica Program.

See Defining Parameters.

Predefined Parameters

The Informatica adapter has the following predefined Parameters:

- **Bulk Load**. Set this Parameter to Yes if you use bulk loading in the Informatica workflow. Oracle Life Sciences Data Hubsupports bulk loading of data only for the staging data processing type. The system drops all indexes on the staging Tables and recreates them after job execution, when you set this Parameter to Yes. See Staging Processing for more information on this data processing type.
- **Drop and Recreate Index**. If set to Yes, the system drops all indexes on all target Table instances before the Oracle Life Sciences Hub Informatica Program is executed, and recreates them after execution. If you do not want to recreate indexes for all the target Table Descriptors, you can call an Oracle Life Sciences Hub API that allows selective index management. See Selective Index Management.
- **Recover Workflow**. If set to Yes, Oracle Life Sciences Hub recovers a suspended Informatica workflow using the Informatica recover mechanism. If set to No, Oracle Life Sciences Hub aborts a suspended Informatica workflow and restarts it.
- **WF Name**. This is the name of the Informatica workflow that you want Oracle Life Sciences Hub to execute. You must provide this name at the time of submitting the Oracle Life Sciences Hub Informatica Program's Execution Setup. See Creating, Modifying, and Submitting Execution Setups.

Selective Index Management

Use the Oracle Life Sciences Data Hub public API for selective index management to:

- Select the target Table instances and the indexes/constraints that you want to recreate.
- Control index management at runtime, as opposed to before and after execution (through the Drop and Recreate Indexes Parameter).

The API has the following signature:

```
CDR_PUB_EXE_RUNTIME.ActOnIndex(Create/Drop:<target_Table_instance_name>:<index/
constraint name>)
```

Call this API from a Stored Procedure Transformation in your Informatica mapping by passing the following values to the Stored Procedure:

- **Create/Drop**. Enter either Create or Drop.
- **Target Table Instance Name**. Enter the Oracle Life Sciences Hub target Table instance name whose index or constraint you want to drop or recreate.
- **Index/Constraint Name**. Enter the index or constraint name.

For example:

```
CDR_PUB_EXE_RUNTIME.ActOnIndex(Create:T_EMPLOYEE:BMP1)
```

> **Note:**
>
> Refer to the *Oracle Life Sciences Data Hub System Application Programming Interface Guide* for more information on Oracle Life Sciences Hub public APIs.

Adding Planned Outputs

No Planned Outputs are required for Informatica Programs.

The system allows you to create them but they have no effect.

See Defining Planned Outputs.

Informatica Integration

This section contains information on the following:

- Informatica Folder Creation
  Oracle Life Sciences Data Hub first creates a folder in Informatica when you install the Program for the first time and subsequently for each check out of the Program.
- Informatica Security Configuration
  This section contains the following topics:

Informatica Folder Creation

Oracle Life Sciences Data Hub first creates a folder in Informatica when you install the Program for the first time and subsequently for each check out of the Program.

The Informatica folder that Oracle Life Sciences Hub creates when you first install the Program is useful only if you already have Informatica mappings and workflow that you want to deploy. For all subsequent interactions with Informatica, Oracle Life Sciences Hub uses the Informatica folder that it creates at the time of checking out the Program.

Informatica Folder Format: When you install the Oracle Life Sciences Hub Informatica Program for the first time,Oracle Life Sciences Hub creates an empty folder in Informatica with a name in this format:

```
LSHProg_<Oracle LSH Program ID>_<Oracle LSH Program Version>
```

For example, for an Oracle Life Sciences Hub Program with the ID `Prg098765`, the corresponding folder created in Informatica's PowerCenter Designer is: `LSHProg_Prg098765_1`.

See Informatica Security Configuration for information on creating Informatica mappings and workflows in the Informatica folder.

Informatica Security Configuration

This section contains the following topics:

- Informatica Security Configuration on Checkout
  When you check out an installed Oracle Life Sciences Data Hub Informatica Program, the following takes place:
- Informatica Security Configuration on Checkin
  When you check in an Oracle Life Sciences Data Hub Informatica Program, the corresponding Informatica folder is locked for write access.
- Informatica Security Configuration on Launching the IDE
  When you launch the PowerCenter Designer from an Oracle Life Sciences Data Hub Informatica Program:

Informatica Security Configuration on Checkout

When you check out an installed Oracle Life Sciences Data Hub Informatica Program, the following takes place:

- Oracle Life Sciences Hub creates another empty folder in Informatica with a name in the same format as at the time of installation with the version number incremented by one:

  `LSHProg_<Oracle LSH Program ID>_<Oracle LSH Program Version>`

- Informatica associates this folder with a security group with the same name as the folder, replacing the prefix `Folder` with `Group`. For example, for the Oracle Life Sciences Hub Program with the ID `Prg098765`, the security group in Informatica is: `Group_Prg098765_1`.
- The Informatica admin user LSHAdmin owns this Informatica security group.

> **Note:**
>
> Each version of the Oracle Life Sciences Hub Informatica Program instance results in a new folder in Informatica. You have access to only the latest Informatica folder.
>
> Also note that although Informatica supports versioning, the Oracle Life Sciences Hub Informatica Repository is nonversioned. You must access versioning information for Oracle Life Sciences Hub Informatica Programs from within Oracle Life Sciences Hub: go to the Actions drop-down list on the Program Properties screen and select View Version History.

Click **Launch IDE** to start Informatica's PowerCenter Designer.

Informatica Security Configuration on Checkin

When you check in an Oracle Life Sciences Data Hub Informatica Program, the corresponding Informatica folder is locked for write access.

However, all users continue to have read access to the Informatica folder. The user who checks out the Program from Oracle Life Sciences Hub will get write access to the Informatica folder.

Informatica Security Configuration on Launching the IDE

When you launch the PowerCenter Designer from an Oracle Life Sciences Data Hub Informatica Program:

- Oracle Life Sciences Hub adds each user who checks out the Oracle Life Sciences Hub Informatica Program to the Informatica security group. This user gets read/write privileges to the Informatica folder. Note that when a user checks out an Oracle Life Sciences Hub Informatica Program and launches Informatica, the user's read/write privileges from all other Informatica folders are taken away. This is because a user can work on only one Informatica folder at a time even if the user has privileges on other folders.

- You can edit mappings and workflows in Informatica only if you launch Informatica from an Oracle Life Sciences Hub Informatica Program that you checked out. You have read-only privileges in Informatica if:
  - Someone else has the Oracle Life Sciences Hub Informatica Program checked out
  - The Oracle Life Sciences Hub Informatica Program is checked in

Defining Oracle Business Intelligence Publisher Programs

An Oracle Life Sciences Data Hub BI Publisher Program lets you use data from Oracle Life Sciences Hub Tables to create reports using Oracle BI Publisher. You can run the Oracle Life Sciences Hub BI Publisher Program from within Oracle Life Sciences Hub. The system generates a report in the desired output format(s) for Oracle Life Sciences Hub Consumers.

See also Setting Up Integrated Development Environments (IDEs)

This section contains the following topics:

- Integration with Oracle BI Publisher
  This section includes information on the process of integration between Oracle Life Sciences Data Hub and Oracle BI Publisher:
- About Oracle BI Publisher Program Source Code
- About Oracle BI Publisher Program Planned Outputs
  BI Publisher supports many output formats.
- Setting Oracle BI Publisher Program Parameters
  Oracle Life Sciences Data Hub BI Publisher Programs include the following types of Parameters.

Integration with Oracle BI Publisher

This section includes information on the process of integration between Oracle Life Sciences Data Hub and Oracle BI Publisher:

- Performing Oracle Life Sciences Data Hub Tasks
- Performing Oracle BI Publisher Tasks
- Running the Program
  Create an Execution Setup for the Oracle Life Sciences Data Hub BI Publisher Program and run it.
- Editing an Existing Program
  When you make changes to the Table Descriptors in an Oracle Life Sciences Data Hub BI Publisher Program:

Performing Oracle Life Sciences Data Hub Tasks

- **Oracle LSH Creates Planned Outputs and Predefined Parameters**. Oracle Life Sciences Hub automatically adds to an Oracle Life Sciences Hub BI Publisher Program definition a Planned Output for each report output format that Oracle BI Publisher supports, and two predefined runtime Parameters.
  See Setting Oracle BI Publisher Program Parameters and About Oracle BI Publisher Program Planned Outputs.

- **Prepare the Program**. You must create and map Table Descriptors in the Oracle Life Sciences Hub BI Publisher Program and install it to enable launching Oracle BI Publisher.
  See Defining Table Descriptors and Installing Program Instances for instructions.
- **Check Out the Program**. Before you launch Oracle BI Publisher, make sure you check out the Oracle LSH BI Publisher Program, so that Oracle Life Sciences Hub can synchronize the Program with the changes you make in Oracle BI Publisher.

> **Note:**
>
> The Oracle Life Sciences Hub Program is equivalent to an Oracle BI Publisher *report*.

Performing Oracle BI Publisher Tasks

Oracle Life Sciences Data Hub creates an Oracle BI Publisher report with the same name as the Oracle Life Sciences Hub BI Publisher Program definition and places the report in a folder under **My Folders**. The name of this folder is in this format:

```
<Program Definition Name>_<Program's obj_id>_Ver<Program's Version No>
```

Edit the Oracle BI Publisher report as follows:

1. **Create a new Data Model and select the Data Source**. The Data Source has a name in the format `LSH_DataSrc_<LSH_application_username>`. Select the Data Source that has your Oracle Life Sciences Hub application username in it. Select the **Only Use Default Schema** checkbox before creating a query. Use the BI Publisher Query Builder to create a query to fetch data from Oracle Life Sciences Hub Tables.

   > **Note:**
   >
   > See "Setting Up Security for Oracle Business Intelligence Publisher" in the *Oracle Life Sciences Data Hub System Administrator's Guide* or contact your Oracle Life Sciences Hub System Administrator if you cannot find a Data Source name with your application username in it.

2. **Create a new layout**. Create a report template in any of the formats that Oracle BI Publisher supports and save the template with this report. Set the Output Format to All Formats to enable Oracle Life Sciences Hub to support all report formats when running the Oracle Life Sciences Hub BI Publisher Program.
3. **Save the report and exit Oracle BI Publisher.** After you exit Oracle BI Publisher, check in the Oracle Life Sciences Hub BI Publisher Program in Oracle Life Sciences Hub. Oracle Life Sciences Hub creates a Source Code definition and instance and uploads the zipped BI Publisher report into the Source Code. See About Oracle BI Publisher Program Source Code.

> **Note:**
>
> Refer to the (*BI Publisher Administrator's and Developer's Guide* and the *BI Publisher Report Designer's Guide*) for complete details. You can browse through the documentation online and download what you need from the Oracle Technology Network. Use this hyperlink to go to the list of available documentation for the Oracle Business Intelligence Suite Enterprise Edition (version 10.1.3.4) http://download.oracle.com/docs/cd/E10415_01/doc/nav/portal_booklist.htm

Running the Program

Create an Execution Setup for the Oracle Life Sciences Data Hub BI Publisher Program and run it.

See Creating, Modifying, and Submitting Execution Setups.

When you execute this Program, the system internally calls BI Publisher APIs to create the report as designed in Oracle BI Publisher.

You can see the final report by going to the My Home tab in Oracle Life Sciences Hub and clicking the Oracle Life Sciences Hub BI Publisher Program's Job ID. See "Tracking Job Execution" in the *Oracle Life Sciences Data Hub User's Guide*.

Editing an Existing Program

When you make changes to the Table Descriptors in an Oracle Life Sciences Data Hub BI Publisher Program:

- You must reinstall and check out the Program after making these changes.
- When you launch Oracle BI Publisher after these changes, Oracle BI Publisher gets the latest tables per your changes in Oracle Life Sciences Hub, but if you want to change the query, you must do that in Oracle BI Publisher.
- When you return to Oracle Life Sciences Hub after saving the report in Oracle BI Publisher, you must check in the Oracle LSH BI Publisher Program.

Do not change anything else in the Oracle Life Sciences Hub BI Publisher Program from within Oracle Life Sciences Hub.

About Oracle BI Publisher Program Source Code

> **Note:**
>
> Do not change the Oracle Life Sciences Data Hub BI Publisher Program Source Code in Oracle Life Sciences Hub. The system creates and updates the Source Code automatically.

Source Code Creation: After you save the Oracle BI Publisher report and exit Oracle BI Publisher (that you launched from Oracle Life Sciences Hub for the first time), you

must check in the Oracle Life Sciences Hub BI Publisher Program. The system creates a Source Code definition and instance and uploads the zipped report file into the Source Code.

Source Code Updates: After every Oracle BI Publisher launch, the system automatically updates this zipped report file when you check the Oracle Life Sciences Hub BI Publisher Program in.

> **Note:**
>
> Oracle Life Sciences Hub marks the latest zipped report file in the Oracle Life Sciences Hub BI Publisher Program's Source Code as Primary. The system disregards any other report files that may be present in the Source Code definition, and uses only this Primary Source Code.
>
> See About Source Code for details on Oracle Life Sciences Hub Source Code.

About Oracle BI Publisher Program Planned Outputs

BI Publisher supports many output formats.

When you create anOracle Life Sciences Data Hub Program of the BI Publisher adapter type, the system automatically adds a Planned Output for each supported output format to the Oracle Life Sciences Hub BI Publisher Program definition. This makes it possible to select any of the supported output formats when running the Oracle Life Sciences Hub BI Publisher Program.

> **Note:**
>
> Make sure that for each Planned Output, Error if Generated and Error if Not Generated are both set to False. See Defining Planned Outputs for more information on Planned Outputs.
>
> Do not change anything else in the system-generated Planned Outputs.

Setting Oracle BI Publisher Program Parameters

Oracle Life Sciences Data Hub BI Publisher Programs include the following types of Parameters.

- Predefined Parameters
  An Oracle Life Sciences Data Hub BI Publisher Program has the following predefined runtime Parameters:
- User-Defined Parameters
  For each user-defined parameter you create in Oracle BI Publisher, create a Parameter in Oracle Life Sciences Data Hub with the same name but of VARCHAR2 data type.

Predefined Parameters

An Oracle Life Sciences Data Hub BI Publisher Program has the following predefined runtime Parameters:

- **BIP Report Output Format**. You may set a default output format from the list of values for this Parameter.
  You can reset the output format at the time of submitting the execution setup. See Creating, Modifying, and Submitting Execution Setups.
- **BIP Template**. This refers to the layout template that you attach to a BI Publisher report. You must type out the layout template's name that you defined in BI Publisher. The template's name does not contain extension names. Do not change anything else in this Parameter.
  You can change the name at the time of submitting the execution setup. See Creating, Modifying, and Submitting Execution Setups.

> **Note:**
>
> Do not change any other properties of these Parameters except the default values, or the Program's execution will fail.

User-Defined Parameters

For each user-defined parameter you create in Oracle BI Publisher, create a Parameter in Oracle Life Sciences Data Hub with the same name but of VARCHAR2 data type.

Installing Program Instances

You can install a Program instance directly from its Properties screen, using the Install button, or in its Work Area (see Installing a Work Area and Its Objects). If you are working with an integrated development environment (IDE) you must install the Program instance in order to see source data in the IDE.

When you install a Program instance using the **Install** button on its Properties screen:

- The system checks in the Program instance and definition, and also the Table instances in the current Work Area to which the instance is mapped.
- The system checks if the Program is installable. If not, the system performs Automatic Mapping by Name on any unmapped target Table Descriptors. If the Program is still not installable and there are still unmapped target Table Descriptors, the system creates Table instances in the current Work Area from the target Table Descriptors and maps them.
- The system attempts to install the Program instance and its source and target Table instances in the current Work Area. The system displays a success or error message. If the installation fails, the error message displays the name of any objects that were not installable.

> **Note:**
>
> If any of the Table instances or the Program definition is not installable, the system cannot install the Program instance. See Installation Requirements for Each Object Type for the reasons these objects may not be installable.

> **Note:**
>
> To continue working on the Program, check it out. It is possible to work on a Program's source code with the Program definition checked in, but the changes do not take effect until you check out the Program definition, upload the source code, and install the Program instance.

- Log File
  To see the log file for the installation, you must go to the Work Area Installation screen.

Log File

To see the log file for the installation, you must go to the Work Area Installation screen.

1. Click the **Applications** tab. The main Application Development screen opens.
2. Click the name of the Work Area you are working in. The Work Area screen opens.
3. From the **Actions** drop-down list, select **Installation History**.
4. Click **Go**. The system displays the Installation History screen with the log files in chronological order.
5. Click the **View Log** link for the most recent installation attempt or for the date and time that you ran the install process. The system displays the log file.

For information on installation and on reading the log file, see Installing a Work Area and Its Objects.

IDE Launch Settings

This section contains the following topics.

- About Launch Settings
  You can work on Oracle Life Sciences Data Hub Programs from an integrated development environment (IDE) which connects to the Oracle database hosting Oracle Life Sciences Hub and retrieves data required for the Program.
- Setting the Blind Break Value
  This setting is relevant only when one or more source Table instances either currently or formerly contained blinded data (whose Blinding Flag is set to Yes).
- Setting the Shared Snapshot Label Value
  If all the relevant Table instances share one or more snapshot labels, those snapshot labels appear in this drop-down list and you can select one.

About Launch Settings

You can work on Oracle Life Sciences Data Hub Programs from an integrated development environment (IDE) which connects to the Oracle database hosting Oracle Life Sciences Hub and retrieves data required for the Program.

To view data as you are developing an Oracle Life Sciences Hub Program in an IDE, do the following before you launch the IDE:

- Define and map the source Table Descriptors you need.
- Install the Program instance; see Installing Program Instances.
- Specify the Launch Settings for blinding and data currency. See Setting the Blind Break Value and Setting the Shared Snapshot Label Value.

Default launch settings are determined by the Data Currency and Blind Break values in the default Execution Setup and your privileges. For example, if the Blind Break setting in the default Execution Setup is Real (Blind Break) but you do not have Blind Break privileges on the Table instances mapped to the Program's source Table Descriptors, your only Blind Break option is Dummy.

> **Note:**
>
> You must have Read Data privileges on the source Table instances to be able to see Dummy data, if you do not have blinding-related privileges.

If there is no Execution Setup defined, the default Blind Break value and options are defined by the blinding status of the Table instances and your privileges, and the default Data Currency value is Current. The values you set here apply only during the current session. See Modifying an Execution Setup and Setting Parameters for information on setting the Data Currency and Blind Break system Parameters in the Execution Setup.

> **Note:**
>
> Launch Settings do not apply to statically referenced Table instances. A statically referenced Table instance is mapped to a source Table Descriptor of a Program containing a Source Code shared to the Program you are working on in the IDE; see Creating and Using Static Reference Source Code.

> **Note:**
>
> If you generate an output on real blinded or real unblinded data, you need additional privileges to see the output. See "Blinding-Related Security Privileges" in the *Oracle Life Sciences Data Hub Implementation Guide* for more information.

Setting the Blind Break Value

This setting is relevant only when one or more source Table instances either currently or formerly contained blinded data (whose Blinding Flag is set to Yes).

Special privileges are required to view real blinded or real unblinded data in these Table instances. You must have these special privileges on all such Tables, in order to see real data in any of them.

> **Note:**
>
> You must have Read Data privileges in order to see any data at all.

The following choices are available depending both on the state of the data and on your security privileges:

- **Not Applicable**. If none of the data has ever been blinded, the only option available is **Not Applicable**. No special privileges are required.
- **Dummy**. This is the only option available to you if you do not have blinding-related privileges for blinded Tables. You can also see this option if you have blinding-related privileges. In that case, you can select this option to work with dummy (not real) data in the IDE.
- **Real (Blind Break)**. If any of the data is currently blinded, and you have the required privileges, you can select this option to view real data in the IDE, according to your company's policies.

  > **Note:**
  >
  > Blind Breaks are not allowed in **SAS Connected Mode With Work Area Data**. Therefore, if you select Real (Blind Break) in **SAS Connected Mode With Work Area Data**, you cannot see any data in SAS. See Connecting to SAS.

- **Real (Unblinded)**. If a blinded Table instance has now been unblinded, you can see real data for the Table instance, provided you have the required privileges. If there are more than one such Table instances, you need the required privileges for all of them to be able to use this option.

Setting the Shared Snapshot Label Value

If all the relevant Table instances share one or more snapshot labels, those snapshot labels appear in this drop-down list and you can select one.

In addition, you normally have the option to view the current data. Your options may be limited by the settings in the default Execution Setup.

You can apply snapshot labels to all the Table instances that a Program reads from or writes to, when you submit its Execution Setup. You can also apply snapshot labels in the Work Area; see Adding, Removing, or Moving a Snapshot Label.

For more information on what snapshots are, see Data Snapshots.

Modifying Programs

If you have the necessary privileges, you can modify a Program either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area.

In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Program Instance Properties
  On the Program instance's Properties screen, click **Update** to enter changes. Oracle Life Sciences Data Hub creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.
- Modifying Program Definition Properties
  You can go to a Program definition's Properties screen in one of the following ways:

Modifying Program Instance Properties

On the Program instance's Properties screen, click **Update** to enter changes. Oracle Life Sciences Data Hub creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must reinstall the Program for the changes to take effect.

You can modify the following:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.

- Definition Source: This field applies to the instance only. It specifies the Program definition to which this Program instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Table Descriptors, Source Code, Parameters, and Planned Outputs.
  - Any new Table Descriptors are not mapped.
  - The Program's status changes to **Non Installable**. If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.

Modifying Program Definition Properties

You can go to a Program definition's Properties screen in one of the following ways:

- **From the Program's Properties screen**: Click the hyperlink of the Program definition that appears in the Definition field.
- **From the Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Program definition screen, click **Update** to enter changes. Oracle Life Sciences Data Hub creates a new version of the definition. You can change the following properties:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information. You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.
- Modifying Table Descriptors
  Table Descriptors belong to the Program definition, but Table Descriptor mappings belong to the Program instance.
- Modifying Source Code
  Source Codes belong to the Program definition. You must check out the definition to add, remove, or update Source Codes.
- Modifying Parameters
  Parameters belong to the Program definition.
- Modifying Planned Outputs
  Planned Outputs belong to the Program definition.

Modifying Table Descriptors

Table Descriptors belong to the Program definition, but Table Descriptor mappings belong to the Program instance.

You must check out the definition to add, remove, or update Table Descriptors, but not to map, unmap, or remap Table Descriptors.

If you need to change a Table Descriptor's columns, you must update the Table Descriptor's definition source either to a different Table definition that meets your needs, or to a new version of the same Table definition, after modifying the Table definition. If you do not have the necessary privileges to modify the source Table

definition, you can probably copy the original Table definition, paste it into the current Application Area, modify it as necessary, and use it as the new definition source.

In the Program instance you can map the Table Descriptor to a different Table instance. See Mapping Table Descriptors to Table Instances for further information.

Modifying Source Code

Source Codes belong to the Program definition. You must check out the definition to add, remove, or update Source Codes.

If a Source Code object has a value in its **Shared From** column, it is an instance of a Source Code definition in another Program. You can modify only a few of its properties; see Source Code Instance.

If a Source Code object does not have a value in its **Shared From** column, it's definition was created in this Program. You can modify all its properties here; see Source Code Definition.

Source Code Instance

You can modify the Source Code name, description, definition source, order, SAS file reference name (if a SAS Program), instance subtype and instance classifications. You can upgrade to a different version of the Source Code definition; see Upgrading Source Code And Undoing Source Code Upgrades.

If you have the necessary privileges, you can go to the Program definition that contains the Source Code definition (it is listed in the **Shared From** column) in the Definitions subtab and modify it there, creating a new version. You must then change the definition source for the Source Code instance in this Program to the new version.

Source Code Definition

If the Source Code definition was created in this Program (in which case there is no entry in the **Shared From** column) the Source Code definition is located in this Program definition, and you can modify it here.

You can edit the actual source code, either in the Editor box or by modifying the file in its development environment (such as SAS) and then uploading it again. You can also change the other Source Code definition properties: File Type, Sharable, Subtype, and classifications.

When you save your changes to a sharable Source Code definition, you have the option to find all instances of the original sharable Source Code and decide whether or not to update them to the new version of the Source Code. See Upgrading One or More Instances from the Definition for further information.

Modifying Parameters

Parameters belong to the Program definition.

You must check out the definition to add, remove, or update Parameters. See Defining Parameters for information.

You can also change some Parameter values and settings in Execution Setups. Select **Execution Setups** from the **Actions** drop-down list in the Program instance in the Work Area. See Creating, Modifying, and Submitting Execution Setups.

Modifying Planned Outputs

Planned Outputs belong to the Program definition.

You must check out the definition to add, remove, or update Planned Outputs. See Defining Planned Outputs for further information.

You can change Planned Outputs' classifications, which affect the classifications of the actual outputs. See Classifying Outputs for further information.

Setting Up Integrated Development Environments (IDEs)

See also IDE Launch Settings

This section contains the following topics:

- Setting Up Oracle SQL Developer or SQL*Plus as an IDE
  To use either Oracle SQL Developer or SQL*Plus to edit and compile your Oracle Life Sciences Data Hub PL/SQL Programs, do the following on your local PC:
- Setting Up SAS as an IDE
  To use SAS as an integrated development environment (IDE), do the following on your local PC:
- Setting Up Informatica as an IDE
  To use Informatica IDE for creating mappings and workflows from within Oracle LSH, do the following on your local PC:

Setting Up Oracle SQL Developer or SQL*Plus as an IDE

To use either Oracle SQL Developer or SQL*Plus to edit and compile your Oracle Life Sciences Data Hub PL/SQL Programs, do the following on your local PC:

- Get the CD-ROM that contains the files **cdrconfig.xml** and **cdrclient.exe** from your system administrator and insert it into your PC. InstallShield automatically runs setup.exe, that loads cdrconfig.xml and cdrclient.exe to a location you specify on your local computer.
- Ensure that **cdrconfig.xml** has the correct directory path for the Oracle SQL Developer or SQL*Plus executable.
- Launching Oracle SQL Developer
  When you click **Launch IDE** from an Oracle Life Sciences Data Hub PL/SQL Program, the following takes place:
- Relaunching Oracle SQL Developer from within Oracle Life Sciences Data Hub
- Relaunching Oracle SQL Developer from outside Oracle Life Sciences Data Hub
  If you log in to Oracle SQL Developer from outside Oracle Life Sciences Hub, you do not get access to the source tables but you can edit the PL/SQL package.

Launching Oracle SQL Developer

When you click **Launch IDE** from an Oracle Life Sciences Data Hub PL/SQL Program, the following takes place:

- If yourOracle Life Sciences Hub PL/SQL Program contains a Source Code instance with a PL/SQL package in it, then Oracle Life Sciences Hub compiles your PL/SQL package.

  > **Note:**
  >
  > If there are any bugs in your PL/SQL package, Oracle SQL Developer fails to launch and an error message related to the bug appears on the Oracle Life Sciences Hub screen.

- Oracle Life Sciences Hub launches Oracle SQL Developer. Connect to the Oracle Life Sciences Hub database using your database user credentials.

  > **Note:**
  >
  > If you cannot find an Oracle Life Sciences Hub database connection to which you can connect, you may have to set up the connection. Contact your System Administrator for more information.

- Table instances mapped to your Program's source Table Descriptors appear as *synonyms* in Oracle SQL Developer; for example, if the Oracle Name of a source Table descriptor is `DEMOG` then there will be a synonym by the name `DEMOG` in Oracle SQL Developer.
- Table instances mapped to your Program's target Table Descriptors appear as empty *tables* in Oracle SQL Developer.
  You can edit, compile, and execute your PL/SQL package in Oracle SQL Developer and these tables reflect the results of your data manipulation. However, the data is not written to Oracle Life Sciences Hub Table instances.

  > **Note:**
  >
  > If you do not want to lose changes you made to the PL/SQL package in Oracle SQL Developer, you must upload your package back into the Oracle Life Sciences Hub PL/SQL Program's Source Code instance before you exit Oracle SQL Developer. See Modifying Source Code for instructions.

Relaunching Oracle SQL Developer from within Oracle Life Sciences Data Hub

If you relaunch Oracle SQL Developer from within Oracle Life Sciences Hub:

- Oracle Life Sciences Hub overwrites the PL/SQL package in Oracle SQL Developer with the PL/SQL package contained in the Source Code instance of the Oracle Life Sciences Hub PL/SQL Program.
- Oracle Life Sciences Hub overwrites any tables that exist in your database schema with the same name as Oracle Life Sciences Hub Tables mapped to the Oracle Life Sciences Hub PL/SQL Program.

Relaunching Oracle SQL Developer from outside Oracle Life Sciences Data Hub

If you log in to Oracle SQL Developer from outside Oracle Life Sciences Hub, you do not get access to the source tables but you can edit the PL/SQL package.

Setting Up SAS as an IDE

To use SAS as an integrated development environment (IDE), do the following on your local PC:

- Get the CD-ROM that contains the files **cdrconfig.xml** and **cdrclient.exe** from your system administrator and insert it into your PC. InstallShield automatically runs setup.exe, that loads cdrconfig.xml and cdrclient.exe to a location you specify on your local computer.
- Install SAS on your PC in the location specified by your system administrator. The location must match the directory path specified in **cdrconfig.xml**.
- Ensure that **cdrconfig.xml** has the correct directory path for the SAS executable.
- Set the user preference for the SAS connection mode. See Connecting to SAS.
- Install any software required to support the preferred connection mode Connecting to SAS.

Setting Up Informatica as an IDE

To use Informatica IDE for creating mappings and workflows from within Oracle LSH, do the following on your local PC:

- Get the CD-ROM that contains the files **cdrconfig.xml** and **cdrclient.exe** from your system administrator and insert it into your PC. InstallShield automatically runs setup.exe, that loads cdrconfig.xml and cdrclient.exe to *`Drive`*`:\Program Files\Oracle\CDR`.
- Install the Informatica client on your PC in the location specified by your system administrator. The location must match the directory path specified in **cdrconfig.xml**.
- Ensure that **cdrconfig.xml** has the correct directory path for the Informatica executable.
- Create a system Environment Variable in Windows with the name INFA_DOMAINS_FILE and set its value to the full path of the `domains.infa` file; for example:

  ```
  INFA_DOMAINS_FILE=drive:\Informatica\PowerCenter8.1.1\domains.infa
  ```

- Set up a user Data Source Name (DSN) named **LSHModel** for your Oracle LSH database account. Oracle LSH imports source and target Tables from your database account into the Informatica folder using this DSN.
  Consult Microsoft Windows online help for instructions on setting up ODBC Data Source Names.

6

Defining Variables and Parameters

See Defining Table Columns for information on defining Columns.

This section contains the following topics:

- About Variables, Parameters, and Columns
- Defining Variables
  Oracle LSH creates Variables automatically based on the Oracle columns or SAS variables that you load or upload from an external system. In addition, you can create Oracle LSH Variables manually, either directly or as part of the process of creating a Table Column or a Parameter.
- Using the Variable Properties Screen
  This section contains the following topics.
- Defining Columns
- Defining Parameters
- Using the Parameter Properties Screen
- Setting Up Parameter Value Propagation
  You can set up Parameter value propagation only in the context of a Workflow or Report Set.
- Defining and Using Parameter Sets
- Defining Programatically Generated Lists of Values and Value Validation
- Modifying Parameters

About Variables, Parameters, and Columns

In the Oracle Life Sciences Data Hub (Oracle LSH) a Variable is the definition source for both Table Columns and Parameters. Its attributes include data type, length, name, default value, and Nullable (Yes/No). These attributes form the basis of both Columns and Parameters; both Columns and Parameters are instances of Variables. Both have additional attributes. Parameters also serve as definitions for Parameter instances, making the only three-level definitional relationship among Oracle LSH objects.

You can use the same Variable as the definition source for multiple Columns and Parameters. This promotes consistency and compatibility: for example, if you base the Study Column of all Tables in a single data flow on the Study Variable, and the Study Parameter of all the Programs that read from and write to those Tables on the same Variable, the Columns and Parameters have the same data type and length, and Nullable setting.

The Nullable attribute, whose default value is Yes in a Variable, is expressed in opposite ways in the user interface for Columns and for Parameters. When you see a Variable whose Nullable attribute is set to Yes through a Column, you see a Nullable setting whose default value is Yes, like the Variable's. When you see the same Variable through a Parameter definition, you see a Required setting whose default

value is No. This is the same attribute setting expressed in different ways: if a Variable is Nullable, it is not Required; if it is Required, it is not Nullable.

Defining Variables

Oracle LSH creates Variables automatically based on the Oracle columns or SAS variables that you load or upload from an external system. In addition, you can create Oracle LSH Variables manually, either directly or as part of the process of creating a Table Column or a Parameter.

This section includes the following topics:

- Creating Variables Automatically
- Creating Variables Manually
- Modifying Variables

Creating Variables Automatically

Oracle LSH creates Variables automatically based on equivalent metadata in external systems in two basic ways:

- Creating Variables through Load Sets
- Creating Variables by Uploading SAS Data Sets and Variables during Table Definition

Creating Variables through Load Sets

Oracle LSH creates Variables automatically when you define or run Load Sets to load data from an external system. These Variables have the same data type and length and other attributes as the external metadata on which they are based.

You can use these Variable definitions within the Application Area where they are created. You can also copy or move them into a Domain library for the purpose of reuse in other Application Areas, according to your organization's policies.

Oracle LSH creates Variables from Load Sets as follows:

- **SAS**. When you define a SAS-type Load Set, you can choose to create the Load Set's Table Descriptors from the SAS Data Sets you want to load. Oracle LSH creates a Table definition for each SAS data set and an Oracle LSH Variable for each SAS variable. The Oracle LSH Variables and Table definitions are located in the same Application Area as the Load Set definition.
- **Oracle Tables and Views**. When you define a Load Set of type Oracle Tables and Views, you can choose to create the Load Set's Table Descriptors from the Oracle tables or views you want to load. Oracle LSH creates a Table definition for each external Oracle table or view and a Variable for each external column. The Variables and Table definitions are located in the same Application Area as the Load Set definition.
- **Oracle Clinical Data Extract**. When you define an Oracle Clinical Data Extract (DX) View Load Set (either Oracle or SAS), you can choose to upload the Load Set's Table Descriptors from the DX Views you want to load. Oracle LSH creates a Table definition for each DX View and a Variable for each Column. These

Variables and Table definitions are located in the same Application Area as the Load Set definition.

- **Oracle Clinical Global Library**. When you run a Global Library Load Set, Oracle LSH creates a Domain with the same name as the Oracle Clinical Domain whose Global Library you are loading (if it does not already exist) and converts all Oracle Clinical Questions to Oracle LSH Variables. If a Question is assigned a Discrete Value Group (DVG) in Oracle Clinical, Oracle LSH also creates a Parameter based on the Variable and gives it a list of values corresponding to the DVG values. Oracle LSH also creates a Table definition from each Question Group.

Creating Variables by Uploading SAS Data Sets and Variables during Table Definition

When you create a new Table in Oracle LSH, you have the option to upload the metadata of a SAS data set, including its variables. When you add a Column to an Oracle LSH Table, you have the option to upload a SAS variable. In both cases, the system converts each SAS variable to an Oracle LSH Variable.

You have the same options when you create a Table Descriptor, which is a Table instance contained in a Program or other executable object.

Creating Variables Manually

Every time you create a Column (in a Table Definition) or Parameter (in a Program, Report Set, or Workflow definition or directly in a Parameter Set definition) you have the option to manually create a Variable or to select an existing Variable as the definition source of the Column or Parameter.

> **Note:**
>
> To promote consistency and compatibility, use existing Variables as often as possible. See Defining and Using Parameter Sets for more information.

When you click **Create LSH Variable** in either the Create Parameter or Create Column screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Description**. See Creating and Using Object Descriptions.
   - **Data Type**. Variables can have a data type of VARCHAR2, NUMBER, or DATE.
   - **Length**. The maximum number of bytes or characters of data that the Column can hold. The requirements vary according to the data type:
     - **VARCHAR2**. A value for length is required and must be between 1 and 4000 characters.
     - **DATE**. The system disregards the length value, if any.

  - **NUMBER**. A value for length is optional.You can leave the length and precision null, and LSH treats the number column as having the maximum possible length.

- **Precision**. (This field appears only if you select a data type of NUMBER.) The total number of digits allowed, not including the decimal marker or a positive (+) or negative (-) sign. Its maximum value is 38.

> **Note:**
>
> Oracle LSH terminology differs from standard Oracle terminology, in which this attribute is called Scale.

- **Oracle Name** (up to 30 characters, uppercase, no spaces). Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces, truncated to 30 characters if necessary.
- **SAS Name**. The SAS Name is used during execution by SAS technologies. The SAS Name can contain up to 32 characters.

  If the value entered for the Name is 32 characters or less, the system uses it as the default value for the SAS Name.
- **SAS Label**. Enter text or accept the default value. The system automatically creates the default from the text you entered in the Name field, converting it to uppercase, with underscores (_) substituted for spaces.
- **SAS Format**. The SAS Format is used during execution by SAS technologies. SAS rules apply.
- **Default Value**. This value becomes the default value of Columns and Parameters based on this Variable. The default is null.
- **Nullable**. If set to Yes, Columns based on this Variable can contain a null value and Parameters based on this Variable are not Required. If set to No, by default Columns based on this Variable must contain a value in each row and Parameters based on this Variable must have a value at execution time. The default value is Yes.

  You can change the Nullable value for Columns and the Required value for Parameters based on this Variable.

2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Parameter or Column.

Modifying Variables

To modify a Variable, do the following:

1. In the Application Area or Domain where the Variable is located, click **Manage Definitions**. The system opens the Maintain screen for the Application Area or Domain.
2. Expand the node (+) for Variables.
3. Click the hyperlink of the Variable you want to modify. The LSH Variable screen opens.
4. Click **Check Out**. The system opens the Check Out screen.

> **Note:**
>
> If the Variable is already checked out, you cannot check it out.

5. Enter a reason for change in the **Comment** field and click **Apply**. The system checks out the Variable.
6. Click **Update**. The system makes the fields editable.
7. Modify as necessary. You can modify all fields.
8. Click **Apply**. The system saves your changes.

If you wish, you can update one or more Parameters that reference this Variable to reference the new version by using the **Upgrade All Instances** function in the **Actions** drop-down list. See Upgrading One or More Instances from the Definition.

Using the Variable Properties Screen

This section contains the following topics.

- Definition Properties
- Buttons

Definition Properties

You can click **Update** to modify any of these properties.

- **Name:** See Naming Objects.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Data Type:** Variables can have a data type of VARCHAR2, NUMBER, or DATE. See Creating Variables Manually for more information.
- **Length:** You can change the length of a VARCHAR2 or NUMBER Variable. See Creating Variables Manually for more information.
- **Precision:** See Creating Variables Manually for more information.
- **Oracle Name:** See Creating Variables Manually for more information.
- **SAS Name:** See Creating Variables Manually for more information.
- **SAS Label:** See Creating Variables Manually for more information.
- **SAS Format:** This is the SAS representation of the data type for this Variable. See Creating Variables Manually for more information.

- **Default Value:** This field shows the default value, if any, for the Parameter definition that refers to this Variable.
- **Nullable:** This field shows whether the Parameters of this Variable can contain null values or not. See Creating Variables Manually for more information.
- **Validation Status:** This field displays the current validation status of the Variable. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Variable. A Variable is always Installable. See Installation Requirements for Each Object Type.
- **Checked Out Status:** This field displays the status of the Variable: either Checked Out or Checked In. See Understanding Object Versions and Checkin/Checkout for further information.
- **Checked Out By:** This field displays the username of the person who has the Variable checked out.
- **Latest Version:** If set to **Yes**, this is the latest version of the Variable definition.
- **Version:** This field displays the current version number of the Variable.
- **Version Label:** This field displays the version label, if any, for the Variable version.

Buttons

- **Check Out:** Click the **Check Out** button to check out the Variable. The button is grayed out if the Variable is already checked out.
- **Update:** Click the **Update** button to modify a Variable. The button is grayed out if the Parameter is not checked out or checked out by someone else.

Defining Columns

Defining Columns is covered in Defining Table Columns in the chapter on Tables.

Defining Parameters

See also:

- Setting Up Parameter Value Propagation
- Defining and Using Parameter Sets
- Modifying Parameters

This section contains the following topics:

- About Parameters
- Creating a Parameter
- Defining Parameter Details

About Parameters

In Oracle LSH you must create a defined object called a Parameter for each input and output parameter in a Program's source code.

In Workflows and Report Sets you can define Parameters especially for the purpose of passing their value (to be set in the Execution Setup definition or at runtime) to other Parameters you specify within the Workflow or Report Set, reducing the number of Parameters to be set at runtime. You can also pass the value of an output Parameter of one Program in a Workflow or Report Set to the input value of another Parameter in the same Workflow or Report Set (see "Setting Up Parameter Value Propagation").

> **Note:**
>
> A Load Set, Data Mart, and Business Area contains Parameters defined by adapters used to create these objects. See Setting Load Set Parameters, Setting Data Mart Parameter Values, and Setting Business Area Attributes and Parameters for more details.

A Parameter definition inherits attribute values from its source Variable (notably data type, length, and default value, if any) and has additional attributes such as Required, Visible, and Read Only, and its list of acceptable values, if any.

- Using Parameters
- Parameters and Execution Setups

Using Parameters

You can use Parameters to increase the reusability of a Program; for example, if you hardcode a value in source code and then want to change the value, you must create a new version of the Program definition and revalidate it. No revalidation of the Program definition is required when you rerun the same Program using a different Parameter value.

Your company may develop a library of standard Parameters that are compatible with standard Table Columns and available for reuse throughout Oracle LSH. Using standard Parameters and Table Columns facilitates reusing Programs on different data and streamlines Parameter value propagation in Workflows and Report Sets.

For example, you can create a Parameter Set containing Parameters based on the same Oracle LSH Variables as the Columns in each standard Table, so that when you search for a Parameter definition in a Program that will read from or write to a standard Table, you can search in the Parameter Set with the same name as the Table.

Some more examples of Parameter usage:

- **Run on different data**. Define a standard study demography report Program definition and promote it to a Domain library. Create instances of this Program in multiple Application Areas, each of which contains data for a different study. Bind the value of the Study Parameter in the Execution Setup for each Program instance to the appropriate Study Parameter value.

Or create a required Parameter settable at runtime to determine whether a Program uses local lab range definitions, standard published reference ranges, or panic ranges.

Similarly, use a Parameter to determine whether you run a Program on all adverse events or only serious adverse events; or use a Parameter to determine whether you run a Program on all Oracle Clinical Data Clarification Forms (DCFs) or only those that have been outstanding for a certain time period.

- **Generate different report formats**. Use a Parameter to generate multiple reports on the same data, each in a different format. For example, create a Parameter called Format with the values Table, Listing, and Figure. Write the source code so that if the input value of the Format Parameter is table, the program produces a table, and so on. If you want to include each format in a Report Set, but want each report to be in a separate Report Set Entry, you can define an instance of the same Program definition in each of three Report Set Entries and bind the Format Parameter to a different value in the Execution Setup of each Program instance.
- **Run on a subset of data**. Use Parameters for subsetting data; that is, limiting the rows processed in a single Table. For example, from a single table that includes data on both male and female patients, you might choose to report data separately, processing only patients whose Sex Parameter value is Male or only patients whose Sex Parameter value is Female. To run the same Program both ways automatically, create two instances of it in a Workflow or Report Set and bind the value in the Execution Setup of each instance so that each instance uses a different value for the Sex Parameter.

Parameters and Execution Setups

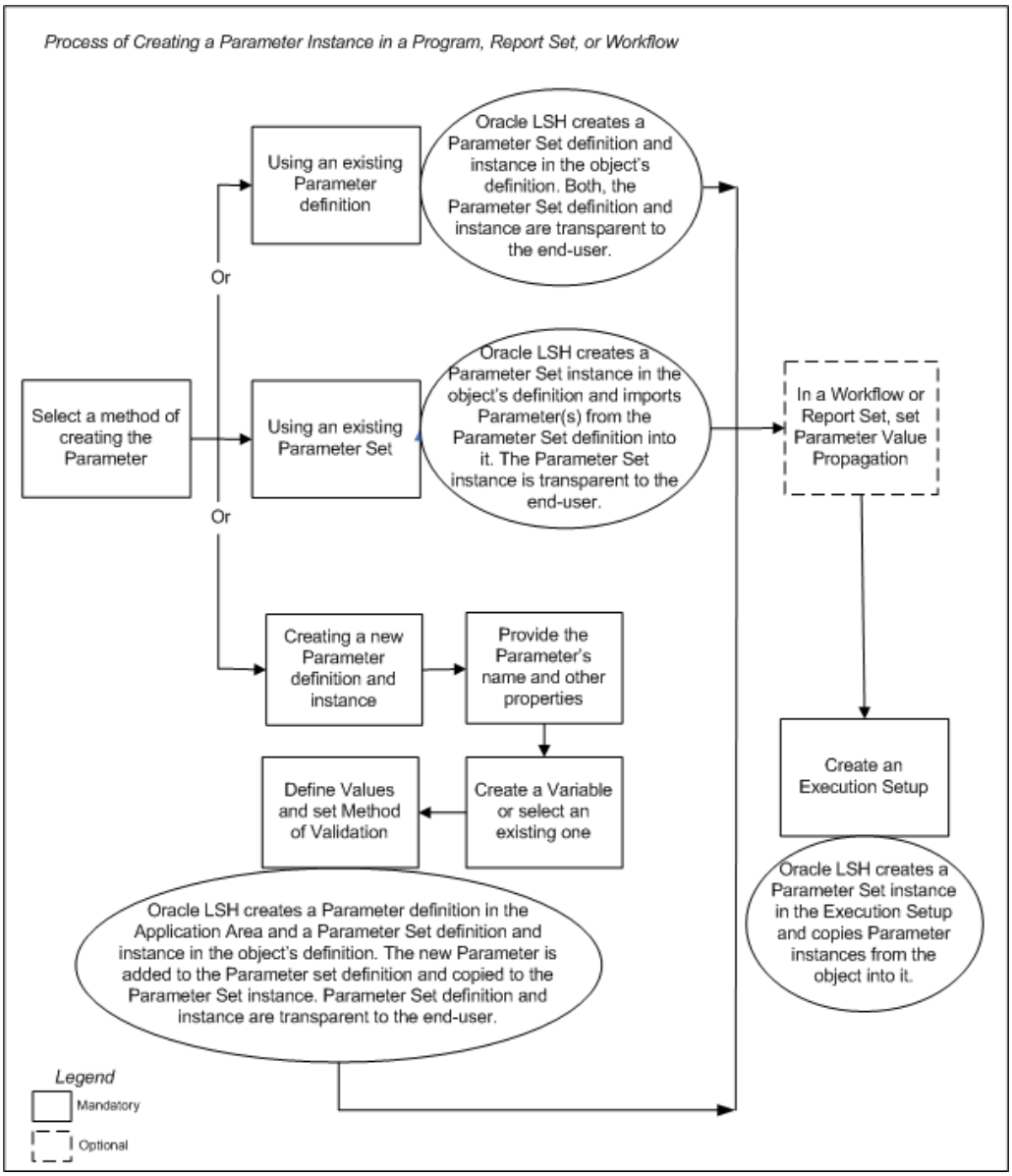
When you generate a default Execution Setup for a Program, the system automatically includes a copy of all the Parameters you have defined.

You can modify Parameter settings in the Execution Setup to control the options available to the person submitting the Program for execution; for example, if a Parameter is set to Read-Only in the Execution Setup, a user submitting the Program for execution cannot change the value of that Parameter. Execution Setups contain a copy of the Parameters and changes made to Parameters in Execution Setups do not affect Parameter settings in a Program definition or a Parameter definition.

You can define multiple Execution Setups, each with different Parameter settings; see Creating, Modifying, and Submitting Execution Setups.

Creating a Parameter

Figure 6-1 Process of Creating the First Parameter Instance in a Program, Report Set, or Workflow



When you create a Parameter in a Program, Report Set, or Workflow, you are actually creating an instance of a Parameter definition. Follow the instructions below.

When you create a Parameter in an Application Area or Domain, you are creating a Parameter definition only, but you can follow the instructions at Create a New Parameter Definition and Instance.

To create a new Parameter instance:

1. In the **Parameter** subtab of a Program, Report Set, or Workflow, click **Add**.

   The system displays the Create Parameter screen.

2. Choose one of the following options:

   - **Create a new Parameter definition and instance**. Choose this option if no Parameter definition exists that can meet your needs.
   - **Create an instance from an existing Parameter definition**. Choose this option if a Parameter definition already exists that meets your needs. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.

     If you use an existing Parameter as a definition source, its source Variable and default values, list of allowable values, and/or validation rules are already defined.
   - **Create Parameters from an existing Parameter Set**. This option creates instances of all the Parameters in a single Parameter Set at once. This option appears only the first time you create a Parameter in a Program, Report Set, or Workflow. That is because an object can contain only a single Parameter Set. See Defining and Using Parameter Sets for more information on Parameter Sets.

     > **Note:**
     >
     > In principle, it is best to reuse Parameter definitions or their source Variables as often as possible. The last two options both reuse existing Parameter definitions. There are two advantages:
     >
     > – promotes data type and length consistency along the data flow when Table Columns are also based on the same Variable
     >
     > – makes automatic Parameter value sharing in Workflows and Report Sets easier to set up

3. Depending on your choice, follow one of the following sets of instructions:

   - Create a New Parameter Definition and Instance
   - Creating an Instance of an Existing Definition
   - Create Parameters from an Existing Parameter Set

- Create a New Parameter Definition and Instance
- Create Parameters from an Existing Parameter Set

Create a New Parameter Definition and Instance

To create a new Parameter definition and instance at the same time, do the following:

1. Select **Create a New Parameter Definition and Instance**. Additional fields appear.
2. Enter values in the following fields:
   - **Name**. See Naming Objects.

> **Note:**
>
> Do not use spaces in the name of any Parameter you create for use in a Report Set. This will cause an error in post-processing because the Parameter name becomes an HTML tag, and spaces are not allowed.

- **Description**. See Creating and Using Object Descriptions.

3. Set the following attributes:
   - **Visible** (Required). If set to **Yes**, the Parameter is visible and its value can be changed. If set to **No**, it will not be visible at runtime and is therefore effectively bound to the value set in the Execution Setup definition.
   - **Required** (Required). If set to **Yes**, the Parameter must have a value; the system will not execute the Program if this Parameter does not have a value. If set to **No**, the system executes the Program even if the Parameter does not have a value.
   - **Read Only** (Optional, and only Visible Parameters can be Read Only). If set to **Yes**, the person submitting the Program for execution can see the Parameter's value but cannot change it. If set to **No**, the submitter can change the Parameter's value at runtime.
   - **Input Output** (Required). From the drop-down list, choose one of the following values: Input, Output, or Input/Output, to define the role of the Parameter in the Program. If set to **Input**, the Parameter can receive a value at runtime and pass it to the Program's source code. If set to **Output**, the Parameter value is generated by the Program and can be passed to another Program in the same Workflow or Report Set. If set to **Input/Output**, the Parameter can receive a value at runtime and pass it to the Program's source code, which may transform the value before reporting it.
   - **Prompt** (Optional). Use this field to specify the label to be displayed in the Execution Setup for this Parameter. If you do not specify a Prompt, the system uses the Parameter Name.

   > **Note:**
   >
   > All the above attributes except Input/Output can be changed during the definition of the Execution Setup.

4. Specify the **LSH Variable** to be used as a definition source for the Parameter. You can select an existing Variable or create a new one. See About Variables, Parameters, and Columns and Creating Variables Manually.
5. Define the allowed and default values of the Parameter. See Defining Allowed Values.
6. Define the validation rules for the Parameter. See Setting Validation Rules.
7. Click **Apply**. The system returns you to the Parameters subtab of the Program Properties screen. To continue defining the Parameter, click its hyperlink. Follow instructions to Defining Parameter Details.

Create Parameters from an Existing Parameter Set

You can create multiple Parameters at the same time by creating instances of all the Parameters in a Parameter Set. This option appears only the first time you add a Parameter to a Program or other executables. See Defining and Using Parameter Sets.

To create a new Parameter definition from an existing Parameter Set, do the following:

1. Select **Create Parameters from an existing Parameter Set**. Additional fields appear.
2. Click the Search icon for the Definition Source field to locate the Parameter Set that contains the Parameters you need. The Search and Select screen opens.
3. In the drop-down lists, select the Domain and Application Area in which to search for a Parameter Set.

   If you know the exact name of the Parameter Set, you can enter it in the Name field.
4. Click **Go**. The system lists all Parameter Sets that meet the search criteria.
5. Click the Quick Select icon to select one Parameter Set. The system returns to the Create Parameter Instance screen.
6. Click **Apply**. The system returns you to the Parameters subtab of the Program Properties screen and, in the Program, creates instances of all the Parameters in the Parameter Set.

If necessary, you can remove Parameters you do not need by selecting them and clicking **Remove**.

To continue defining the Parameter, click its hyperlink. Follow instructions for Defining Parameter Details.

Defining Parameter Details

See also Setting Up Parameter Value Propagation.

This section contains the following topics:

- Setting a Method of Determining Value
- Defining Allowed Values
- Setting Validation Rules

Setting a Method of Determining Value

If you are defining a Parameter in the context of a Workflow or Report Set, you have a choice between the following methods of determining the Parameter value. You must choose one.

If you are defining a Parameter for a Program that is not part of a Workflow or Report Set, getting the Parameter's value from another Parameter is not possible and this choice does not appear.

- **Define Value**. The Parameter's value must be set manually; either here, or in the Execution Setup, or at runtime. You can specify one of several ways to determine allowable value(s). See Defining Allowed Values.
- **Get Value From Another Parameter**. You can set up value propagation within a Workflow or Report Set so that this Parameter receives its value from another Parameter in the same Workflow or Report Set. See Setting Up Value Propagation from the Target Parameter.

Defining Allowed Values

You can specify the values allowed for the Parameter in several ways, found in the **List of Values** drop-down list. The system displays the appropriate interface for defining each after you make a choice.

- **None**. No list of allowable values is associated with the Parameter. The user can enter any value. You can define a Program to validate the value; see Defining Programatically Generated Lists of Values and Value Validation.

  You can specify a default value. To force the user to set a value at runtime, select Required and do not set a default value.

  The system validates that the default value conforms to the Parameter's data type and length (defined in the source Variable). If the data type is Date, the system checks that it is in the format specified in User Preference.
- **Static List of Values**. Create a list of specific values you want to allow for the Parameter. These values will appear as the only possible choices for the Parameter. Click **Add Value** to enter each value you want to allow. The system will display them in the list of values in the order in which they appear here.

  You can select one value to be the default value if you want to. To force the user to set a value at runtime, select **Required** and do not set a default value.

  The system validates that each value you define conforms to the Parameter's data type and length (defined in the source Variable).

  > **Note:**
  >
  > Do not enter a comma (,) in the Parameter value string. The system interprets a comma as a delimiter between two different Parameter values.

- **Classification List of Values**. This option results in a list of allowable values that comprises the values at a particular level of a classification hierarchy.

  Select a classification hierarchy and level. The system generates a list of values equal to the set of values defined for that hierarchy level. For example, if you choose the Project, Study, and Site hierarchy and the Study level, the list of values includes all the Study names.

  If you define two Parameters in the same Program and base their allowed values on the same classification hierarchy, on two adjacent levels, the allowable values for the Parameter associated with the lower level are limited to those with a relation to the value chosen for the Parameter associated with the higher level.

  For example, if you define two Parameters whose possible values are linked to the Project, Study, and Site hierarchy, with one associated with the Project level and

the other associated with the Study level, the possible values for the Project level Parameter include all the Project values in the hierarchy. However, the possible values for the Study level Parameter are limited to those related to the particular Project value selected. If the value for the Project-level Parameter is Project 1, the list of values that appears for the Study-level Parameter includes only those studies that are part of Project 1. If you change the value of the Project level Parameter to Project 2, the list of values for the Study level Parameter changes to studies that are part of Project 2.

The same is true for Parameters linked to each lower level. In this example, you can define a third Parameter associated with the Site level whose values are limited to sites that are part of the particular study selected.

- **Program Generated List of Values**. You can define an Oracle LSH Program, or Source Code within a Program, specifically for the purpose of dynamically generating a list of values for a Parameter. You must check in and install the Program.

  You must write the source code in PL/SQL and use the following syntax to generate a list of values:

  ```
  FUNCTION <function name> (pi_paramRef IN CDR_PARAMETER_OBJ_TYPE,
  pi_paramColl IN CDR_PARAMETER_COLL)
  RETURNS CDR_VALS_COLL;
  ```

  Click the **Search** icon and specify the Program and Source Code you want to use to generate the list of values for the Parameter.

  See Defining Programatically Generated Lists of Values and Value Validation for further information.

Setting Validation Rules

You can choose a method of validating the Parameter's value. If you select a method, the system validates the Parameter value when the Program is submitted for execution. If the Parameter has an invalid value, the system returns an error message and does not execute the Program.

> **Note:**
>
> If you set a default value, the system validates its data type and length each time you modify the Parameter.

You can choose one of the following methods of validating the Parameter's value:

- **None**. You can choose to accept any value entered by the user.
- **Programmatic**. You can define an Oracle LSH Program specifically for the purpose of dynamically validating the Parameter value. You must check in and install the Program. See Defining Programatically Generated Lists of Values and Value Validation for further information.

  Click the **Search** icon and specify the Program that contains the Source Code you want to use to validate the value of the Parameter.
- **Validate Against LOV**. If you defined a static list of values for the Parameter, the system accepts the entered value only if it appears on that list.

Using the Parameter Properties Screen

This section contains the following topics.

- About the Parameter Properties Screen
- Instance Properties
- Define Values
- Validate Values
- Definition Properties
- Variable Properties
- Buttons

About the Parameter Properties Screen

You can reach this screen from Oracle LSH object definitions, instances, and Execution Setups. Click **Update** to make changes to any of the settings but note the following:

- Changing Parameter properties from an object's instance is the same as changing Parameter properties from an object's definition. This is because Parameters belong to an object definition.
- The changes you make to the Parameter properties become the default settings for all new Execution Setups for this object instance. But any existing Execution Setups do not get updated with the changed Parameter properties until you upgrade those Execution Setups.
- If you are in an Execution Setup, the changes you make to the Parameter apply only to the Execution Setup.

See Modifying Parameters for information on modifying Parameter properties.

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Visible:** If set to **Yes**, the Parameter is visible by default in the Execution Setup of the instance. If set to **No**, the Parameter is not visible by default. You can choose to keep this setting as **Yes** in an object definition or instance, and change it whenever required, in an Execution Setup.

> **Note:**
>
> Changes made to Parameter properties from an Execution Setup do not affect Parameter properties in the object's definition because Execution Setups maintain an independent copy of the Parameters.

- **Required:** If set to **Yes**, you must provide a value for this Parameter in the Submission form for this instance. This setting is applicable only to visible Parameters. You can choose to keep this setting as No in an object definition or instance, and change it when required, in an Execution Setup.
- **Read Only:** If set to Yes, you cannot change the value of this Parameter. You can choose to set this property to **No** in an object definition or instance, and change it whenever required, in an Execution Setup.
- **Input/Output:** This setting indicates the type of Parameter. Parameters can be of type Input, Output, or Input/Output.
- **Prompt:** This is the display prompt for the Parameter. By default it is the same as the Parameter's name.
- **Definition:** You can upgrade to a new version of the same definition. See Upgrading to a Different Definition Version from an Instance.
- **Version:** This field displays the current version number of the Parameter instance.

  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.
- **Validation Status:** This field displays the current validation status of the Parameter instance. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Parameter: Installable or Non Installable. The installable status of a Parameter depends on the installable status of its parent object. See Installation Requirements for Each Object Type for more details.

Define Values

This section displays the following settings:

- **List of Values:** If you have provided a method to determine the Parameter's value, a list of values that are valid for the Parameter appear here. Else the value for this field is None.
- **Default Value:** If you have specified a default value for the Parameter, the value appears here.

  See Defining Allowed Values for more details on defining values for Parameters.

Validate Values

- **Method of Validating Values:** If you have specified a method to validate Parameter values, that method name appears here. See Setting Validation Rules for more details.

Definition Properties

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. All Parameter properties belong to the Parameter definition as well as the Parameter instance. You can modify all properties from the Parameter instance whether or not the Parameter definition is checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Latest Version:** If set to **Yes**, this Parameter instance is pointing to the latest version of the Parameter definition. If set to **No**, this Parameter instance is pointing to an older version of the Parameter definition.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Validation Status:** This field displays the current validation status of the Parameter definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Parameter in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Load Set: Installable or Non Installable. See Installation Requirements for Each Object Type.

Variable Properties

This section displays the properties of the Variable that the Parameter definition belongs to.

- **Data Type:** This field displays the data type of the Variable: VARCHAR2, NUMBER, or DATE.
- **Oracle Name:** This field displays the Oracle name of the Variable to which the Parameter definition belongs.
- **Length:** This field displays the length of VARHCAR2 or NUMBER type Variables.
- **SAS Name:** This field displays the SAS name of the Variable.
- **SAS Label:** This field displays the SAS label of the Variable.
- **SAS Format:** This field displays the SAS format of the Variable.

Buttons

Update: Click the **Update** button to modify the Parameter instance. See Modifying a Parameter Instance.

Setting Up Parameter Value Propagation

You can set up Parameter value propagation only in the context of a Workflow or Report Set.

This section contains the following topics:

- About Parameter Value Propagation
- Setting Up Value Propagation from the Source Parameter
- Setting Up Value Propagation from the Target Parameter

About Parameter Value Propagation

Within a Workflow or Report Set, you can set up automatic Parameter value propagation so that a Parameter contained in a Program instance in the Workflow or Report Set can receive its value automatically from another Parameter in the same Workflow or Report Set; either the output Parameter of another Program, or a Parameter created directly in the Workflow, Report Set, or Report Set Entry especially for this purpose.

Value propagation has two benefits:

- **Consistency**. Value propagation ensures a consistent value among the Parameters you specify. For example, many Programs in a Report Set may have an input Parameter for the Study value. If the whole Report Set concerns only one study, you can set up value propagation from a special Value Propagation Parameter you define at the top Report Set level for this purpose, and all the study Parameters in the Report Set.
- **Ease of Submission**. Value propagation makes it much easier to submit a Report Set or Workflow for submission because all Parameters defined as getting their value from another Parameter cannot be modified, and the person submitting the job is not required to enter a value for them.

You can set up value propagation in the definition of either the source or the target Parameter. If you want the value of a single Parameter to be propagated to multiple other Parameters, it is more efficient to set it up from the source Parameter.

The rules for defining value propagation are slightly different in Workflows and Report Sets.

Value Propagation in Workflows: Because a Workflow has a set execution order, you must always propagate Parameter values from Parameters whose value is entered or generated earlier during Workflow execution to Parameters that require a value later in Workflow execution. That is, the source Parameter must be either defined at the top level of the Workflow or be an input Parameter of a Program executed before the Programs containing the target Parameters to which its value is propagated.

Value Propagation in Report Sets: In Report Sets, the rules for value propagation are different depending on whether the source Parameter is defined directly in the Report Set/Report Set Entry, or is an output Parameter of a Program in a Report Set Entry:

- **Propagating values from Report Set or Report Set Entry Parameters**. If you define a Parameter directly at the top level of the Report Set or in any Report Set Entry especially for the purpose of propagating its value to other Parameters in the Report Set or Report Set Entry, you must define the Parameter at the same level or a higher level from the Report Set Entries that contain the target Parameters.
- **Propagating values from Program output Parameters**. Oracle LSH can adjust the execution order of a Report Set to accommodate the Parameter value propagation you define from the output Parameter of one Program to the input Parameter of another Program. You can define value propagation from a Program

Parameter in a Report Set Entry that is displayed below a target Program Parameter's Report Set Entry. However, the system does not allow circular value propagation.

> **Note:**
>
> If you plan to use a triggered submission for a Workflow or Report Set, you must ensure that each Parameter contained in the Workflow or Report Set receives its value from a Parameter at the top level of the Workflow or Report Set or from an output Parameter generated by a Program within the Workflow or Report Set. For information on triggered submission, see Using Message-Triggered Submission from External Systems.

Setting Up Value Propagation from the Source Parameter

The following kinds of Parameters can serve as the source of the value in Parameter value propagation:

- **Program Output Parameters**. You can use a Parameter value generated by one Program in a Report Set or Workflow to populate the value an input Parameter of one or more other Programs in the same Report Set or Workflow.
- **Report Set and Workflow Parameters**. You can define Parameters especially for this purpose directly in a Report Set, Report Set Entry, or Workflow.
- **Report Set Overlay Template and Post-Processing Parameters**. All Report Set Entries have the same set of predefined Parameters that the system uses to generate a single or multivolume PDF output. These Parameters are predefined to automatically pass their value to Parameters with the same name. You can explicitly set up value sharing for Post-Processing Parameters but not for Overlay Template Parameters; see Creating and Setting Report Set Parameters for further information.

You can set up value propagation either manually or automatically.

- Setting Up Value Propagation Manually
- Setting Up Value Propagation Automatically

Setting Up Value Propagation Manually

To define value propagation from the source Parameter manually, do the following:

1. On the Parameter instance's screen, click the **Update** button. The screen refreshes to make all fields editable.
2. Scroll down to the **Value Propagation** section. Click **Add Parameters**.
3. Choose from a list of Parameters. The system displays only Parameters that are either:
   - contained in the same Report Set or Workflow
   - (in a Report Set only) defined directly in a Report Set Entry or a Program in a Report Set Entry

In a Report Set, it is possible to feed a value from an output Parameter of one Program to an input Parameter in a Program in the Report Set. The system executes the Program that produces the output Parameter before executing the Program that uses the value for an input Parameter. The system prevents a circular reference.

> **Note:**
>
> It is possible to set a Parameter as a source for value propagation even if the target Parameter is not of the same data type or size or both. Such a mismatched value propagation causes the Workflow or Report Set to fail when executed. You must make sure you set up value propagation keeping in mind the data type and size of the source and target Parameters.

Setting Up Value Propagation Automatically

You can set up automatic value propagation to Parameters that may be added in the future to the Report Set or Workflow.

When you add a Program that contains a Parameter with the same name to the Workflow or Report Set, the system automatically sets up value propagation to it from this Parameter, and displays a message informing you that it has done so.

For Report Sets, the system sets up automatic value propagation only to Parameters that are in the direct flow below the value source Parameter.

For Workflows, the system sets up automatic value propagation only to Parameters that occur downstream in the execution flow.

To set this up, select the **Automatically Pass Value to Parameters With Same Name** check box.

Alternatively, if you prefer not to use automatic propagation or if you want to pass the value to Parameters with a different name, you can set up value propagation Malay. See Setting Up Value Propagation Manually.

Setting Up Value Propagation from the Target Parameter

This option is available only for Parameters contained in a Program that is part of a Report Set or Workflow, and for Parameters defined directly in a Report Set Entry. The Parameter's value must be passed to it by one of the following:

- an output Parameter of a different Program in the same Report Set or Workflow
- a Parameter created for this purpose in the Report Set or Workflow
- in the case of post-processing Parameters in Report Sets, by the same predefined post-processing Parameter at the top of the Report Set or Report Set Entry

Do the following to set value propagation from the target Parameter:

1. On the Parameter's screen, click **Update**. The screen refreshes to make all fields editable.

2. Under **Method of Determining Value**, select **Get Value From Another Parameter**.

   The system displays the **Parameter** field with a **Search** icon.

3. Click the **Search** icon. The system opens a screen listing all the eligible source value Parameters contained in the Report Set or Workflow.

4. Select a **Parameter** by clicking its radio button and clicking the **Select** button. The system enters the value source Parameter's name in the **Parameter** field and its type and location in the fields below. You can click on **Parameter Details** to see the whole definition of the value source Parameter.

> **Note:**
>
> The system uses the Parameter links you set up here to determine the execution order of Report Sets (which is not necessarily the same as the display order). The system prevents you from defining circular links.

Defining and Using Parameter Sets

Defining a set of standard Parameter Sets facilitates creating Parameter instances in Program and other executable instances and facilitates setting up Parameter value propagation in Report Sets and Workflows.

As soon as you add a Parameter to a Program, Report Set, or Workflow, behind the scenes, the system creates a Parameter Set definition and instance in that executable's definition. The Parameter you create really gets added to the Parameter Set definition and instance. You can never see a Parameter Set instance in the user interface, but a Parameter exists exclusively inside a Parameter Set. In addition, each executable can contain only a single Parameter Set. This is why you see the option of creating a Parameter using an existing Parameter Set only once.

When you are working in a Program, Report Set, or Workflow, the simplest way to create multiple Parameter instances is to create instances of all the Parameters in a Parameter Set at the same time. However, you must create logical Parameter Sets in Application Areas or Domains in order to use this functionality.

See Figure 6-1.

If you define the first Parameters by selecting a Parameter Set, then the system creates an instance of that Parameter Set in the executable definition. The Parameter Set definition remains in the Application Area or Domain. This has implications for modifying the Parameters in your executable, as follows:

- **If you have Modify privileges on the Parameter Set definition** in the Application Area or Domain, you can make structural changes to the Parameter Set, but behind the scenes the system checks out the Parameter Set definition, creates a new version of it, and makes the same changes in the new version so that the Parameter Set definition remains consistent with the Parameter Set instance in the executable. The next time someone uses the same Parameter Set definition, he or she will see the modified version as the most current. Structural changes include adding and removing a Parameter and changing the source definition of any Parameter.

- **If you do not have Modify privileges on the Parameter Set definition** in the Application Area or Domain, you cannot add or remove Parameters from your executable, or change the source definition of any Parameter, because the Parameter Set instance and definition must remain synchronized.

Solution 1: To avoid either of these situations and yet benefit from standard Parameter Sets, before you create Parameters in a Program or other executable, copy the Parameter Set definition in its Application Area or Domain and paste it into your local Application Area. Create Parameters in your executable from the local Parameter Set definition. You can then add and remove Parameters and the changes will affect only the local copy of the Parameter Set definition.

Solution 2: Create a version label on the version of the Parameter Set that is the correct, standard version. Always search for Parameter Sets with Version Labels that are not null, and use the correct version.

- Explicitly Defining Parameter Sets

Explicitly Defining Parameter Sets

You may want to take the time to develop standard Parameter Sets based on standard Variables, logically grouped and meaningfully named.

To create a Parameter Set definition manually, do the following:

1. In the Applications tab, navigate to the Application Area or Domain where you want to create the Parameter Set.
2. Click **Manage Definitions**. The system opens the Maintain Application Area (or Domain) Library screen.
3. From the **Create** drop-down list, select **Parameter Set**.
4. Click **Go**. The system opens the Create Parameter Set Definition screen.
5. Enter a **Name** and **Description** of the Parameter Set that will help other users determine whether to use this Parameter Set.
6. Click **Apply**. The system opens the Parameter Set screen.
7. Under Parameters at the bottom of the page, click **Add**. The system opens the Create Parameter screen.
8. Add as many Parameters as necessary. See instructions at Creating a Parameter and Defining Parameter Details.
9. To make the Parameter Set available for use, check it in.

Defining Programatically Generated Lists of Values and Value Validation

You can define Source Code within a Program to generate a list of values for a Parameter, or to validate a user-entered value. You can use the same Source Code for both purposes.

When you search for Source Code to use for a Parameter's programmatic LOV or value validation, the system does not impose the normal security requirements; you can select any sharable Source Code from any Program, whether or not you have standard security access to the Program.

The source code you write must be in PL/SQL and must reference CDR_VALS_COLL, a predefined Oracle database object of type Collection whose structure is effectively the same as a table's, with rows and columns. CDR_VALS_COLL has two columns, one for position (row number in the table) and one for value (the value to derive for the list of values or validation). You must write source code that creates as many positions as necessary and populates each with an appropriate value.

> **Note:**
>
> If the source code generates only one value, the system assigns that value to the Parameter automatically.

CDR_VALS_COLL is effectively empty except when a Program populates it within the context of a particular user's session. Each user session "sees" its own collection only.

See example source code below.

To create a programmatically generated list of values for a Parameter, or to programmatically validate a Parameter value, do the following:

1. Create an Oracle LSH Program of type PL/SQL. See Creating a Program.
2. Add a Source Code definition to the PL/SQL Program. See Defining Source Code and Defining PL/SQL Programs.
3. In the Oracle Package field, enter the package name.
4. In the Oracle Procedure field, enter the function name. You are writing a function, not a procedure, but enter the function name in this field.
5. Add a source Table Descriptor for each Table instance your code will read from, if any. Map each Table Descriptor to the appropriate Table instance. See Defining Table Descriptors and Mapping Table Descriptors to Table Instances .
6. Enter your PL/SQL source code in the Source Code field.

   As with other PL/SQL source code in Oracle LSH, you must include the package spec as well as the package body in your source code.

   Be sure to use the same names for the package and function that you entered in the Oracle Package and Oracle Procedure fields.

   The syntax in Example 6-1 is required verbatim up to the beginning of the function, except for the package and function names. See the examples for further information.
7. Apply your changes, check in the Program definition, and install the Program instance in its Work Area.
8. In the Parameter definition, specify the name of the Program and Source Code that contain the code that will generate the LOV or validate the Parameter value. See Defining Allowed Values and Setting Validation Rules.

Example 6-1 Simple Example

In this example, the source code creates two rows in the collection CDR_VALS_COLL. Position 1 has the value Yes and position 2 has the value No. This example shows the essential elements required in as simple a way as possible. However, you would not

want to use this particular source code because you could accomplish the same thing much more simply by defining a static list of values with two values: Yes and No.

The package spec is the first five lines of code, up to the slash (/). The package name is **LOV** and the function name is **returnValues**. Except for these names, for which you can substitute any names you choose, all the code above the beginning of the actual function (at `begin`) is required as is.

For each row in the collection the source code does the following in succession:

- Creates the row itself by using an EXTEND command
- Specifies the values for that row of the POSITION column
- Specifies the values for that row of the VALUE column
- Specifies that the function returns the value for position *n*

Example source code:

```
CREATE OR REPLACE package LOV as
FUNCTION returnValues (pi_tParam IN cdr_parameter_obj_type ,pi_cParamColl    IN
cdr_parameter_coll) RETURN CDR_VALS_COLL;
end LOV;
/
CREATE OR REPLACE package body LOV as
FUNCTION returnValues (pi_tParam IN cdr_parameter_obj_type ,pi_cParamColl    IN
cdr_parameter_coll) RETURN CDR_VALS_COLL
IS
     valuesRecord     CDR_VAL_OBJ_TYPE  := new CDR_VAL_OBJ_TYPE(NULL,NULL);
     LOVCollection    CDR_VALS_COLL     := CDR_VALS_COLL();

begin

        LOVCollection.EXTEND;
        valuesRecord.POSITION := 1;
        valuesRecord.VALUE    := 'Yes';
        LOVCollection(1) := valuesRecord;

        LOVCollection.EXTEND;
        valuesRecord.POSITION := 2;
        valuesRecord.VALUE    := 'No';
        LOVCollection(2) := valuesRecord;


     RETURN
(LOVCollection);

end returnValues;
end LOV;
/
```

Example 6-2 Pulling Column Values from a Table Instance

In the real world you would use a programmatically generated LOV to pull the current values of a column in an Oracle LSH Table instance that meet a certain set of criteria. In this case, you add a source Table Descriptor to the Program and map it to the Table instance whose Column values you want to read.

In your source code you write a SELECT statement and WHERE clause to describe the criteria for the values you want to retrieve. You then use the EXTEND command to

create *n* rows in the collection, where *n* is the number of values retrieved, and populate each row with a POSITION value and a VALUE value.

Example 6-3 Pulling Values from an External System

You can pull values from a table in an external system if you include the necessary remote location connection information so that the system can read data in the external system.

Modifying Parameters

The types of changes you are able to make to a Parameter definition or instance depend on many different interdependent factors:

- Some Parameter properties belong to the Parameter instance, some to the Parameter definition, and others to the Variable. These three definitional layers may each belong to a different container, and you may have different privileges on each.
- To modify a Parameter within a Program or other executable object, you must have Modify privileges on the Program or other executable object itself, and that object must be checked out.
- A Parameter Set instance must remain structurally the same as its Parameter Set definition. If you add or remove Parameter instances from a Parameter Set instance, the system automatically creates a new version of the Parameter Set definition and removes them there as well. You must have Modify privileges on the Parameter Set definition or you cannot add or remove Parameter instances from the Parameter Set instance, or change the source Parameter definition for Parameter instances within the Parameter Set instance.
- If you are working in an Execution Setup, you can make very few changes to Parameters.

You can make the following changes in the following locations:

Parameter Definition: Parameter definitions are located in two places: contained directly in an Application Area or contained directly in a Domain. In either of these locations, if you have the required privileges, you can do the following:

- Update the properties of the Parameter definition itself.
- Select a different source definition Variable for the Parameter definition.
- Check out the source definition Variable through the Parameter definition, then go to the Variable and modify it. The Parameter definition references the new version of the Variable.

Parameter Instance in a Parameter Set Definition: If you have the necessary privileges, you can go to a Parameter Set definition contained directly in a Domain or Application Area and modify the Parameter instances it contains. You can make the following changes:

- If you have the necessary privileges on the Parameter definition, you check out the Parameter definition through the Parameter instance, the go to the Parameter definition and modify it. The Parameter instance references the new version of the Parameter definition.
- Using the Update button, you can modify the properties belonging to the Parameter instance.

Parameter Instance in a Parameter Set Instance in an Executable Object: Parameter Set instances are located in Programs and other executables and in Execution Setups. The following applies to Parameter Set instances that are located anywhere except in an Execution Setup:

If you created the Parameter Set instance in the executable by creating Parameters based on an existing Parameter Set, the Parameter Set definition is located in an Application Area or Domain. In this case:

- You cannot check out the Parameter definition from a Parameter instance.
- You can select a different source definition Parameter for the Parameter instance only if you have Modify privileges on the Parameter Set definition. This is a structural change, and the system automatically checks out the Parameter Set definition and creates a new version of it with the change.

> **Note:**
>
> Load Sets, Data Marts, Business Areas, and some Program types have predefined Parameter instances whose source definition Parameters are located in Adapter Areas. You should not modify these Parameters, and the administrator should set up security so that it is not possible to modify the Parameter definitions.

If you created the Parameter Set instance in the executable by creating one Parameter at a time—either defining a new Parameter definition and instance, or creating an instance of an existing Parameter—then both the Parameter Set definition and the Parameter Set instance are located in the same executable. In this case:

- You can check out the source definition Parameter from the Parameter instance and make changes as necessary.
- Any structural changes you make—adding or removing Parameters, or selecting a different source definition Parameter for an instance—are automatically applied to the Parameter Set definition as well as the Parameter Set instance.

Parameter Instance in a Parameter Set Instance in an Execution Setup: You cannot update the Parameter instance or check out and modify the source definition Parameter from a Parameter instance contained in an Execution Setup.

- Modifying a Parameter Instance
- Modifying a Parameter Definition

Modifying a Parameter Instance

To modify a Parameter instance in a Program or other executable, do the following:

1. Click the Parameters subtab to view the Parameter instances.
2. Click the hyperlink in the **Name** column of the Parameter you want to modify. The system displays the Parameter Instance screen.
3. Click **Update**. The system displays most fields as updatable.

> **Note:**
>
> You must check out the definition of the Program, Report Set, or Workflow that contains a Parameter instance before you can update its properties. If the executable definition is not checked out, the Parameter instance's **Update** button is not enabled.

4. Modify the settings and/or source definition as necessary.
5. Click **Apply**. The system saves your changes in the Parameter instance only.

Modifying a Parameter Definition

If you have the necessary privileges, you can modify a Parameter definition directly in the Application Area or Domain, or through an instance of the Parameter in a Parameter Set in an Application Area or Domain.

> **Note:**
>
> Never change anything in a Parameter definition from a Load Set, Data Mart, and Business Area. If you do, the adapter you used to create the Load Set, Data Mart, or Business Area, becomes invalid and either you cannot create any more objects of that adapter type, or the objects you create do not work.

- Modifying a Parameter Definition Directly in a Domain or Application Area
- Modifying a Parameter Definition through a Parameter Set

Modifying a Parameter Definition Directly in a Domain or Application Area

To modify a Parameter definition directly in a Domain or Application Area, do the following:

1. Navigate to the Domain or Application Area in the Applications tab.
2. Click the **Manage Definitions** icon. The system opens the Maintain Library screen.
3. Click the Parameter node (+). The system displays all the Parameters in the Application Area library.
4. Click the hyperlink of the Parameter you want to modify. The system opens the Parameter Definition screen.
5. Click **Check Out**. The system checks out the Parameter and creates a new version of it.
6. Click **Update**. The system makes all the fields enterable.
7. Modify the settings as necessary. For an explanation of each setting, see Create a New Parameter Definition and Instance and Defining Parameter Details.

   If you need to change the attributes that belong to the Variable (Data Type, Length, and, for Number types, Precision) you can select a different Variable in the Definition field or, if you have Modify privileges on the source definition Variable,

you can check it out through the Parameter definition and modify it as necessary. The Parameter definition references the new version of the Variable.

8. Click **Apply**. The system saves your changes.

Modifying a Parameter Definition through a Parameter Set

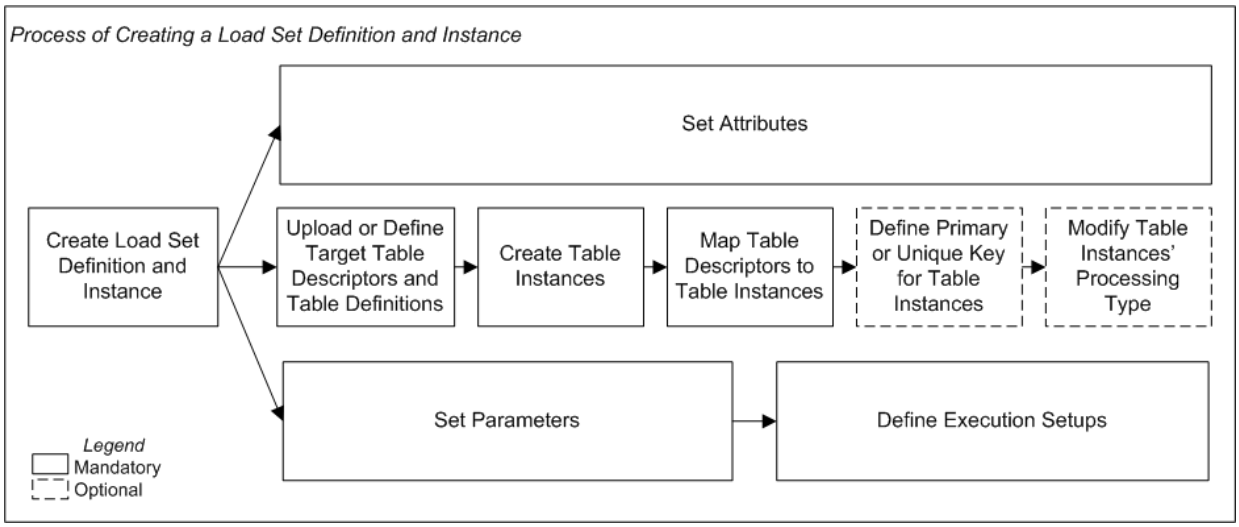
When you modify a Parameter definition through an instance of it in a Parameter Set, the instance automatically points to the new version of the Parameter definition. To modify a Parameter definition through an instance of it in a Parameter Set, do the following:

1. Navigate to the Domain or Application Area in the Applications tab.
2. Click the **Manage Definitions** icon. The system opens the Maintain Library screen.
3. Click the Parameter Set node (+). The system displays all the Parameter Sets in the Application Area library.
4. Click the hyperlink of the Parameter Set that contains the Parameter you want to modify. The system opens the Parameter Set definition screen.
5. Click **Check Out**. The system checks out the Parameter Set and creates a new version of it.
6. Click the hyperlink of the Parameter you want to modify. The system opens the Parameter instance screen.
7. In the Parameter Definition section in the lower part of the screen, click **Check Out**. The system opens the Check Out screen.
8. Enter the reason you are checking out the Parameter definition in the **Comment** field and click **Apply**. The system returns you to the Parameter instance screen.
9. Click the hyperlink to the Parameter definition in the **Definition** field in the Instance Properties section at the top of the screen. The system opens the Parameter Definition screen.
10. Click **Update**. The system makes fields enterable.
11. Modify the settings as necessary. For an explanation of each setting, see Create a New Parameter Definition and Instance and Defining Parameter Details.
    If you need to change the attributes that belong to the Variable (Data Type, Length, and, for Number types, Precision) you can select a different Variable in the Definition field or, if you have Modify privileges on the source definition Variable, you can check it out through the Parameter definition and modify it as necessary. The Parameter definition references the new version of the Variable.
12. Click **Apply**. The system saves your changes.

7

Defining Load Sets

Figure 7-1 Process of Creating a Load Set Definition and Instance



This section contains information on the following topics:

- About Load Sets
- Creating a Load Set
- Using the Load Set Properties Screen
  If you are working in a Work Area, you see the properties of both the Load Set instance and the Load Set definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.
- Defining Table Descriptors
- Setting Load Set Parameters
- About Load Set Planned Outputs
- Defining Different Load Set Types
- Installing Load Set Instances
- Modifying Load Sets

About Load Sets

To load data into the Oracle Life Sciences Data Hub (Oracle LSH), you define and run a Load Set.

Oracle LSH allows you to load data from a wide variety of sources. The standard Oracle LSH installation (without any special customizations) allows you to load data

from SAS, from any Oracle database, or from a text file. Oracle LSH also has a set of Load Set types developed especially for Oracle Clinical that allow you to load data from data extract views, from the discrepancy management subsystem, and from the global library.

Oracle LSH uses a specialized executable defined object called a Load Set to handle data loading. A Load Set definition identifies the type of source data system and includes Table Descriptors that specify the required structure of one or more Tables into which to load the data in Oracle LSH. For Oracle and SAS Load Sets the system can create Table Descriptors based on tables or data sets in the source data system. For all types, you can create Table instances based on Table Descriptors and map them automatically, using a job in the **Actions** drop-down list.

For Load Sets that load files—SAS and Text—you should not specify the file during Load Set definition because the system uploads the actual data file at that time. Let the person who executes the Load Set specify the file to get the most current data. For the same reason it does not make sense to set up a repeating schedule for running a SAS or Text Load Set; you would be loading the same data each time.

For Oracle-technology Load Sets you can specify default values in the Load Set definition for the tables to be loaded. The system only creates views during Load Set definition, and always loads the most current data when you run the Load Set. You can set up a repeating execution schedule to update the data as necessary.

At runtime, if the user submitting the Load Set specifies different external tables or files to load from the ones you used to create the Table Descriptors, the Load Set will run successfully if the data structure is compatible.

For information on loading blinded data, see Loading Real and Dummy Data.

Load Set Types: The different types of Load Sets are as follows. You see only those types that your company uses and that you have security access for.

- Oracle Tables and Views. Oracle LSH loads data and metadata from any Oracle database.
- SAS. Oracle LSH loads one SAS data set at a time or, using a CPORT or XPORT file, multiple data sets. The system loads metadata at the time of Load Set definition and data at Load Set runtime.
- Text. Oracle LSH loads one or more text files into one or more Table instances.
- Oracle Clinical Data Extract Views . Oracle LSH loads patient data from Oracle views defined in Oracle Clinical.
- Oracle Clinical Data Extract SAS Views. Oracle LSH loads patient data from SAS views defined in Oracle Clinical.
- Oracle Clinical Global Metadata . Oracle LSH loads your Oracle Clinical Global Library definitions and automatically converts them to Oracle LSH definitions as follows: converts Oracle Clinical Questions to Oracle LSH Variables. If a Question is associated with a DVG, Oracle LSH also converts the Question to an Oracle LSH Parameter with the DVG as its list of allowed values. Oracle LSH converts Oracle Clinical Question Groups to Oracle LSH Tables with Columns based on the Variables corresponding to each Question in the Question Group.
- Oracle Clinical Labs . Oracle LSH loads Lab definitions, Lab Ranges, and Lab Assignment Criteria, and other Lab-related metadata (see OC Labs Load Set Table Descriptors for a complete list).

- Oracle Clinical Study Data . Oracle LSH loads study-specific data including discrepancies, Data Clarification Forms (DCFs), page tracking information and patient status information.
- Oracle Clinical Stable Interface . Oracle LSH loads tables described in the *Oracle Clinical 4.x Stable Interface Guide*.
- Oracle Clinical Randomization . Oracle LSH loads Oracle Clinical tables related to the Randomization subsystem.

You can customize Oracle Clinical to use the Oracle Messenger facility to trigger a Load Set. For example, you could trigger a patient data Data Extract Load Set with a successful batch validation run, or trigger a Global Library load with a successful replication to remote nodes.

Security: Each Load Set type uses a different predefined Oracle LSH adapter to handle the data and metadata exchange between the external system and Oracle LSH. An adapter consists of a group of predefined objects in an Adapter Area, which is contained in an Adapter Domain.

Before you can create a Load Set for any Oracle source data system you must do the following setup: enter connection information for the remote location into Oracle LSH in the Remote Locations subtab of the Administration tab. Instructions are included in the chapter "Registering Locations and Connections" in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

- Create one or more user groups and assign them to Adapter Areas (see "Setting Up Security for Adapters" in the *Oracle Life Sciences Data Hub System Administrator's Guide*). Each Definer who will need to define Load Sets must belong to a user group that is assigned to the Adapter Area that corresponds to the Load Set type that he or she needs to define.

  In addition, you must add an administrator a user group assigned to each Oracle-technology adapter so that the administrator can define remote locations for the Oracle adapters.
- An administrator must define remote locations for the Oracle adapters. If you want to use shared connections, the administrator must create them. See the chapter on "Defining Remote Locations" in the *Oracle Life Sciences Data Hub System Administrator's Guide* for further information.
- Individual users with access to external Oracle systems can create their own connections for use in defining and running Load Sets Preferences, accessed from a link on the Oracle LSH My Home screen. See "Getting Started" in the *Oracle Life Sciences Data Hub User's Guide* for further information.

> **Note:**
>
> When you define any Oracle technology-based Load Set you must enter a Remote Location and your own (or shared) connection information in the Attributes section of the Load Set definition in order to create the Load Set's Table Descriptors.
>
> However, be careful not to enter your own or shared connection information as a runtime Parameter because this represents a breach of security on the source data system, enabling anyone with security access to the Load Set itself to run the Load Set regardless of whether he or she has the required privileges on the source system.
>
> Leave the Parameter value empty so that the user must enter his or her own connection information to run the Load Set.

Execution: Load Sets, like other Oracle LSH executable objects, have Execution Setups that serve as the basis for the submission form for running the job. You can modify the default value of Parameters in the Execution Setup definition, and the user submitting the Load Set can change them at runtime. See Creating, Modifying, and Submitting Execution Setups.

Oracle LSH uses the Processing Type you define for the target Table instances to determine how to process the data and whether to maintain an internal audit trail of inserts, updates, and deletes (see Data Processing Types for further information).

Loading Blinded Data: See "Loading Real and Dummy Data".

Reports on Load Set Definitions and Instances: From the **Actions** drop-down list, you can generate reports that provide information on a Load Set definition or instance; see System Reports for information.

Creating a Load Set

When you create a Load Set in a Work Area, you are actually creating an instance of a Load Set definition.

To create a new Load Set instance:

1. In a Work Area, select **Load Set** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Load Set screen.
3. Choose one of the following options:
   - **Create a new Load set Definition and Instance**. Choose this option if no Load Set definition exists that can meet your needs, either as it is or with some modification.
   - **Create an instance of an existing definition**. Choose this option if a Load Set definition already exists that meets your needs.

     If you can adapt an existing Load Set definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the

copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.

4. Depending on your choice, follow one of the following sets of instructions:

- Creating a New Load Set Definition and Instance
- Creating an Instance of an Existing Load Set Definition

Creating a New Load Set Definition and Instance

When you select **Create a new Load set Definition and instance** in the Create Load Set screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Description**. See Creating and Using Object Descriptions.
   - **Load Set Type**. See Defining Different Load Set Types for information.
2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Load Set.

   The system opens the Properties screen for the new Load Set instance.

See the following sections for general information that applies to most Load Set types:

- Setting Load Set Attributes
- Setting Load Set Parameters
- Defining Table Descriptors
- About Load Set Planned Outputs

See Defining Different Load Set Types for information about each type of Load Set.

- Setting Load Set Attributes
  Some Load Set types include attributes. These vary depending on the type of Load Set; see the instructions for each type of Load Set. These values cannot be changed at runtime.

Setting Load Set Attributes

Some Load Set types include attributes. These vary depending on the type of Load Set; see the instructions for each type of Load Set. These values cannot be changed at runtime.

Creating an Instance of an Existing Load Set Definition

If you use an existing Load Set as a definition source, all of its Table Descriptors, Parameters and other characteristics are already defined. See Creating an Instance of an Existing Definition for instructions.

After you have created the Load Set instance, you must map the Table Descriptors to Table instances; see Mapping Table Descriptors to Table Instances for instructions.

Using the Load Set Properties Screen

If you are working in a Work Area, you see the properties of both the Load Set instance and the Load Set definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See Modifying Load Sets for information on modifying Load Sets.

See also Figure 7-1.

This section contains the following topics:

- Instance Properties
- Definition Properties
- Load Set Attributes
- Buttons

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Definition:** This field specifies the Load Set definition to which this Load Set instance points.

  You can upgrade to a new version of the same definition. See Upgrading to a Different Definition Version from an Instance.
- **Version Label:** This field displays the version label, if any, for the current Load Set instance version.
- **Validation Status:** This field displays the current validation status of the Load Set instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Load Set: Installable or Non Installable. See Installation Requirements for Each Object Type.
- **Version:** This field displays the current version number of the Load Set instance.

  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.

Definition Properties

You can see the following definition properties:

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to modify Table Descriptors, Parameters, or Planned Outputs. However, you can change Table Descriptor mappings without checking out the definition. See Understanding Object Versions and Checkin/Checkout for further information.
- **Latest Version:** If set to **Yes**, this Load Set instance is pointing to the latest version of the Load Set definition. If set to **No**, this Load Set instance is pointing to an older version of the Load Set definition.
- **Checked Out By:** This field displays the username of the person who has the Load Set definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Load Set/Adapter Type:** This field displays the type of Load Set or Adapter.
- **Validation Status:** This field displays the current validation status of the Load Set definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Load Set in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Load Set: Installable or Non Installable. See Installation Requirements for Each Object Type.

Load Set Attributes

This section of the Load Set screen displays attribute values for the type of Load Set you have created. See Setting Load Set Attributes.

Buttons

From a Load Set instance in a Work Area, you can use the following buttons:

- **Install:** Click **Install** to install the Load Set instance, including mapping target Table Descriptors and installing mapped target Table instances; see Installing Load Set Instances. For a list of reasons a Load Set instance may not be installable, see Installation Requirements for Each Object Type.
- **Submit:** Click **Submit** to run the Load Set instance. Before you can run the Load Set, you must install it and create an Execution Setup for it (select **Execution Setups** from the **Actions** drop-down list).
- **Update:** Click **Update** to modify the Load Set instance properties. See Modifying Load Set Instance Properties. You can also update Load Set Attributes by clicking **Update** in the Load Set Attributes section of the screen. See Setting Load Set Attributes.
- **Check In/Out and Uncheck:** Click these buttons to check out, check in, or uncheck the Load Set definition. Different buttons are displayed in the Load Set Definition Properties section depending on the Checked Out Status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of

the person who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.

Defining Table Descriptors

Load Sets have only target Table Descriptors. They must be compatible with the structure of the external tables or data sets from which you are loading data and with the Table instances into which you are loading data.

For all Load Set types, Oracle LSH can create the Table Descriptors based on data structures you specify in the source system. You can then create Table instances based on those Table Descriptors, or create Table instances based on standard (or any other compatible Table definitions) and map the two. See Mapping Columns of Different Data Types and Lengths for an explanation of compatibility; for example, a source column or variable is compatible with a target column of a longer length, but not with a target column of a shorter length.

The target Table instance must have one of the following process types: For information on the first three processing types, see Data Processing Types.

- **Staging with Audit** (the default)
- **Staging Without Audit**
- **Reload**
- **Pass-Through View** For Oracle-technology Load Sets, you also have the option to create the target Table instances as pass-through views so that the physical source data remains in the external Oracle system. This approach allows you to minimize data storage space. After you map the Table instance to an Oracle-type Load Set, the system adds this option to the Processing Type drop-down list of the Table instance.

  It is not necessary to run Load Sets whose target Table instances are defined as pass-through views. As soon as you install the Load Set, the data in the external Oracle system is available to Oracle LSH.

  Programs can use Table instances defined as pass-through views as sources just as they can Table instances that contain data. When you run the Program, it reads data in the source system and can write data to Oracle LSH Table instances.

Mapping Table Descriptors to Table Instances: You can do either of the following:

- Run the **Table Instances from Existing Table Descriptors** job from the **Actions** drop-down list to create a Table instance of the same structure as the Table Descriptor and automatically map the two. See Creating Table Instances from Table Descriptors and Simultaneously Mapping Them for instructions.
- Create a Table instance—for example, create an instance of a standard CDISC Table definition—and map the Table Descriptor to the Table instance. If they have the same name, you can run the **Automatic Mapping by Name** job from the **Actions** drop-down list and it will map any Columns that have the same name. If not, you can map them manually. See Mapping Table Descriptors to Table Instances .

Enforced Compatibility at Runtime: At runtime a user can specify a remote data source with a different structure from the Table Descriptor you defined. If the structure are incompatible—the differences are such that they would cause data corruption—the load fails with an error.

The following types of differences cause a failure:

- The target Oracle LSH Table Column of a character data type is shorter than the source table column or data set variable.
- For number data types, the length and precision combined are smaller in the target than in the source.
- The target Table Column has a data type that is incompatible with the source data type. See Mapping Columns for information on compatible data types.
- The source table or data set includes a column or variable that is not included in the Table Descriptor.

See the section on each type of Load Set for details:

- Oracle Load Set Table Descriptors
- SAS Load Set Table Descriptors
- Text Load Set Table Descriptors
- OC Stable Interface Tables Load Set Table Descriptors
- OC DX SAS Views Load Set Table Descriptors
- OC Data Extract Views Load Set Table Descriptors
- OC Design and Definition Load Set Table Descriptors
- OC Global Metadata Load Set Table Descriptors
- OC Labs Load Set Table Descriptors
- OC Randomization Load Set Table Descriptor
- OC Study Data Load Set Table Descriptors

For general instructions see Defining and Mapping Table Descriptors.

Setting Load Set Parameters

Each Load Set type has a different set of runtime Parameters. The system uses these Parameters to create the Execution Setup to be used to submit the Load Set for execution.

> **Note:**
>
> Because these are predefined and required Parameters you should not change anything except the default value in the Load Set definition, or the Load Set may not function properly. For Remote Location Parameters you should not even enter a default value for security reasons.

The predefined Parameters are described for each type of Load Set:

- Oracle Load Set Parameters
- SAS Load Set Parameters
- Text Load Set Parameters
- OC Stable Interface Tables Load Set Parameter

- OC DX SAS Views Load Set Parameters
- OC Data Extract Views Load Set Parameters
- OC Design and Definition Load Set Parameters
- OC Global Metadata Load Set Parameters
- OC Labs Load Set Parameters
- OC Randomization Load Set Parameters
- OC Study Data Load Set Parameters

About Load Set Planned Outputs

Oracle LSH automatically creates the following Planned Outputs for Load Sets:

- **Log File**. Oracle LSH creates a log file Planned Output for all types of Load Sets. It is the only Planned Output for Load Sets of Oracle Tables and Views and OC adapters.
- **Error File**. Oracle LSH creates an error file Planned Output for SAS and Text Load Sets and saves each rejected row into this error file. The generation of the error file does not indicate a job failure. A SAS or Text Load Set job fails when the number of rejected records exceeds the value of the Maximum Allowed Errors parameter.
- **Input data file**. For SAS and Text Load Sets, if you set the Save Input File attribute to Yes, the system automatically creates a Planned Output and saves the input file as an actual output at runtime. If you classify this Planned Output, the system classifies the saved input file as an output with the classification you specified.
- **Control File**. Oracle LSH creates this Planned Output for Text and SAS Load Sets. This Planned Output is always generated for Text Load Set jobs. For SAS Load Sets, the system generates it only for bulk loads (when you set the parameter Direct to Yes). This Planned Output stores the SQL*Loader control file.
- **Textload.zip**. Oracle LSH creates this Planned Output if you set the **Save as Input** attribute to **Yes** while uploading a zipped Load Set file. This is applicable only to Text Load sets.

You can classify a Planned Output in order to classify the actual output when it is generated; see Classifying Outputs for further information.

To view the log file or saved input file for a Load Set job, do one of the following:

- Use the Search or Advanced Search feature.
- Browse for the log file in the Outputs subtab of the Reports tab.

Defining Different Load Set Types

This section includes information on the following topics:

- Oracle Tables and Views
- SAS
- Text

- Oracle Clinical Stable Interface
- Oracle Clinical Data Extract SAS Views
- Oracle Clinical Data Extract Views
- Oracle Clinical Design and Definition
- Oracle Clinical Global Metadata
- Oracle Clinical Labs
- Oracle Clinical Randomization
- Oracle Clinical Study Data

Oracle Tables and Views

This section contains the following topics:

- About Oracle Tables and Views Load Sets
- Oracle Load Set Attributes
- Oracle Load Set Table Descriptors
- Oracle Load Set Parameters
- Oracle Load Set Planned Outputs
- Oracle Load Set Execution Setups

About Oracle Tables and Views Load Sets

Oracle Tables and Views Load Sets enable you to access data in Oracle LSH from any Oracle system external to Oracle LSH. The source of the data in the external system can be either a database table or a view with column data types of varchar2, number, and date.

The adapter loads the primary key definitions of the source tables to the target Table instances when you define a Load Set.

You can take two approaches to Oracle data in Oracle LSH:

- **Load the physical data into Oracle LSH**. Rerun the Load Set periodically to refresh the data in Oracle LSH.
- **Define the Load Set's target Table instances as a pass-through view**, so that the physical source data remains in the external Oracle system. This approach allows you to minimize data storage space.

  Programs can use Table instances defined as pass-through views as sources just as they can Table instances that contain data. When you run the Program, it reads data in the source system and can write data to Oracle LSH Table instances.

  You specify whether to use the Load Set to load or view data as part of the definition of the target Table instance to which you map the Table Descriptor. The system adds the **Create Table as a View** option to the **Process Type** drop-down list of the Table instance after you map the Table instance to an Oracle-type Load Set.

> **Note:**
>
> All target Table instances of the Load Set must be defined the same way—either as pass-through views or not. If some are defined as pass-through views and others are not, execution of the Load Set fails.

Oracle Load Set Attributes

Click **Update** and enter values for the remote location and database schema that are the source of the data to be loaded:

- **Remote Location**. Click the Search icon and choose a source remote location/connection combination from the list of values. The system displays only those locations and connections to which you have security access.

  If the location you need is not on the list, ask a system administrator to create a defined Remote Location for it in the Administration user interface.

  If you have a valid username and password for the required database, you can create a connection in the Preferences hyperlink at the top of most Oracle LSH screens. However, you cannot create a connection until an administrator has created a remote location.

  Your company may also use shared connections. If so, you can use those as well.

- **Schema**. Select the database schema on the remote location where the table or view resides.

Oracle Load Set Table Descriptors

Use the **Upload Table Descriptors** function to specify the tables you want to load. The system generates a list of tables in the remote location that you specified in the Attributes section.

See also Defining Table Descriptors.

Oracle Load Set Parameters

The following are Oracle Load Set runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **Direct**. If you set Direct to Yes (default is No), the system uses the direct path INSERT to load data into the target Table instance(s).

> **Note:**
>
> When writing to multiple Table instances, if all the Table instances are not of the Transactional High Throughput processing type, the system first inserts data into the Transactional High Throughput processing type Tables using the direct INSERT and subsequently inserts data using the conventional INSERT into Tables of other data processing types. The Load Set job completes with warnings and writes warning messages to the job log.

- **Drop and Recreate Indexes**. This parameter applies only if you set Direct to Yes.

  If set to Yes (default), the system drops all non-unique indexes before running a data-loading job and recreates the non-unique indexes after loading data into the target Table instances. The system does not drop indexes if Direct is set to No.

- **Logging**. This parameter applies only if you set Direct to Yes.

  If set to Yes (default), in the event of a failure the system can recover data committed to the database. If set to No, the system does not maintain a redo log file and cannot recover any data when a database failure occurs.

Oracle Load Set Planned Outputs

Oracle Load Sets have only one Planned Output: a log file. See About Load Set Planned Outputs for further information.

Oracle Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See Oracle Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

SAS

This section includes the following topics:

- About SAS Load Sets
- UTF8 Encoding
- SAS Load Set Attributes
- SAS Load Set Table Descriptors
- SAS Load Set Parameters
- SAS Load Set Planned Outputs
- SAS Load Set Execution Setups

About SAS Load Sets

A SAS Load Set loads data from a SAS file into Oracle LSH. You can define a SAS Load Set for a single data set or for multiple data sets in a SAS XPORT or CPORT file.

Each time you run a SAS Load Set, you load all the data contained in the source file at runtime. You can load a zipped data set file (but not a zipped XPORT or CPORT file).

UTF8 Encoding

To ensure that Oracle LSH stores and displays special characters in your data correctly, start SAS in UTF8 mode.

If you are using SAS 9.2:

- In Windows:

```
Drive:\Program Files\SAS\SASFoundation\9.2\sas.exe" -CONFIG
Drive:\Program Files\SAS\SASFoundation\9.2\nls\u8\SASV9.CFG"
```

- In UNIX:

```
sas -encoding UTF8
```

If you are using SAS 9.1.3 on UNIX, use sas_dbcs instead of sas to start in unicode mode. For further information, see SAS Paper Paper 1036, *Multilingual Computing with the 9.1 SAS Unicode Server* at: https://support.sas.com/resources/papers/proceedings/proceedings/sugi28/281-28.pdf.

SAS Load Set Attributes

The only attribute is the Save Input File? flag:

- If set to **Yes**, the system saves the uploaded file as an output. See SAS Load Set Planned Outputs.
- If set to **No**, the uploaded file is deleted after the load completes. This is the default value.

SAS Load Set Table Descriptors

If you are using a transport file as a source, you can load more than one data set with a single Load Set, but you cannot load a zipped file. If you are using a data set file as a source, you can load only that one data set but you can load it as a zipped file.

> **Note:**
>
> If you are loading a data set, and the table descriptor associated with its table definition uses a Sort Constraint, to preserve the sort order, make sure that the data set is not empty.

Oracle LSH automatically creates one Table Descriptor for each data set in the file you specify. If you specify a transport file with data sets that have indexes defined, Oracle LSH creates corresponding indexes in the Table definition as well.

To create Table Descriptors for a SAS Load Set, do the following:

1. In the Table Descriptors subtab of the Load Set, click **Add**. The system opens the Create Table Descriptors screen.

2. Select Create New Table Definition(s) and Descriptor(s) from a SAS file. The system refreshes the screen and adds a field with a **Browse** button.
3. Click **Browse**. The system opens a standard Browse window.
4. Navigate to the SAS file you want, select it, and click **Open**. The system returns to the Create Table Descriptors screen and enters the name of the file you selected in the Import From File field.
5. Click **Apply**. The system creates one Table Descriptor for each data set in the file you specify.

> **Note:**
>
> To create a SAS Load Set definition, Oracle LSH loads the file you specify with all its data. The more data the data set contains, the longer it takes to create the Load Set and the more resources are consumed. Therefore, copy the structure of the data set in SAS and use the empty copy to create the Table Descriptors.

You can use the **Create Table Instances from Existing Table Descriptors** job from the **Actions** drop-down list to create and map all the Table instances you need. Alternatively, you can create instances of any compatible Table definitions and map them.

SAS Load Set Parameters

SAS Load Sets have the following runtime Parameters:

> **Note:**
>
> These are predefined, required, runtime Parameters and you should not change any of their properties, or the Load Set may not function properly.
>
> These Parameters take meaningful values only at the time of submitting the Load Set's Execution Setup.

- **BLOB ID (Temporary)**. This Parameter is for internal use only. It contains a pointer to the uploaded file.
- **Dataset File Name**. Do not select a file while defining the Load Set. Oracle LSH uploads the file when you submit the Load Set's Execution Setup. If you upload the file during Load Set definition, the data will probably be outdated at the time the Load Set is executed. Therefore, leave this field blank. The user must enter the filename at runtime if **Load From Server OS** is set to No else the user has to enter the Server OS filename.
- **Load from Server OS**. You have the option to load a data set file from your local computer or from a remote system. Set this parameter to Yes (default is No) if your file is either on the Oracle LSH DP server or on another computer that has its drives NFS-mounted on the DP server computer. If you set this parameter to No,

you must upload a data set file from your local computer at the time of submitting the SAS Load Set job.

> **Note:**
>
> The system can load remote SAS data set files only from an Oracle LSH DP Server that is running the SAS and the SQL*Loader services. In addition, the SAS Service Location must point to the SQL*Loader executable. Provide the absolute path of the SQL*Loader executable in the Details field of the SAS Service. See the chapter on Setting Up Services in the *Oracle Life Sciences Data Hub System Administrator's Guide* for more information.

- **Server OS Filename**. If you set Load From Server OS to Yes, enter the absolute path of the remote data set file at the time of submitting the Load Set's Execution Setup else leave this parameter blank.
- **Maximum Allowed Errors**. Tolerance factor; the maximum number of invalid records you are willing to accept before SQL*Loader stops the loading process and marks the Load Set job as failed. The default value of this parameter is 0.

> **Note:**
>
> If you are uploading a SAS CPORT or XPORT file, note that the value you enter here applies to *each* data set contained within the CPORT or XPORT file.

- **Direct**. If set to Yes (default is No), the system uses the direct path INSERT to load data from the SAS dataset file into the Oracle LSH target Table instance.

> **Note:**
>
> When writing to multiple Table instances, if all the Table instances are not of the Transactional High Throughput processing type, the system uses the direct path INSERT for the Transactional High Throughput processing type Tables and the conventional INSERT for Tables of other data processing types. The Load Set job completes with warnings and writes warning messages to the job log.

- **Drop and Recreate Indexes**. If set to Yes (default), and if the value of the parameter Direct is Yes, the system drops all non-unique indexes before running a data-loading job and recreates the non-unique indexes after loading data into the target Tables. The system does not drop indexes if Direct is set to No.

> **Note:**
>
> See *Oracle® Database Utilities* at https://docs.oracle.com/cd/E11882_01/server.112/e22490/toc.htm for more information on SQL*Loader parameters at:

SAS Load Set Planned Outputs

Oracle LSH automatically creates the following Planned Outputs for SAS Load Sets:

- **SAS Loading Log File**. Oracle LSH writes the job log into this Planned Output for all jobs except bulk loading jobs. Bulk loading jobs write data from SAS data sets into Transactional High Throughput Table instances.
- **SAS Loading Input Data File**. If you set the Save Input File attribute to Yes, the system uses that Planned Output as a placeholder for the input file saved as an output during execution. If you set Save Input File to No, the system still creates the Planned Output but does not create the actual output when you run the Load Set.

  You can classify the Planned Output and view the actual output through the Oracle LSH Outputs user interface, subject to normal Oracle LSH security rules. To view the contents of the file, you must download it and view it through SAS.
- **SAS Bulk Loading Error File**. If you set the parameter Direct to Yes, and if the data loading job encounters one or more invalid records, the system generates this file and writes each rejected record into it. For SAS CPORT and XPORT files, Oracle LSH consolidates the error files that SQL*Loader generates for each data set.
- **SQL*Loader Log File**. If you set the parameter Direct to Yes, SQL*Loader generates this log file for each data set when writing to a Table instance and stores in this Planned Output. For SAS CPORT and XPORT files, Oracle LSH consolidates the log files that SQL*Loader generates for each data set.
- **SQL*Loader Control File**. If you set the parameter Direct to Yes, SAS generates the SQL*Loader control file for each data set when you run the Oracle LSH Load Set job. The system stores the control file in this Planned Output. If you use CPORT or XPORT, Oracle LSH consolidates all the control files into this Planned Output.

SAS Load Set Execution Setups

When you define an Execution Setup for a SAS Load Set, be careful to give it a name and description that will enable a person submitting the Load Set for execution to understand which file should be loaded.

The file must contain data sets that correspond to the Table Descriptors defined for the Load Set, but the Table Descriptors are created as part of the Load Set definition, while the file should only be specified at runtime. Therefore, provide enough information in the Execution Setup Name and Description to make it clear which data sets this Load Set definition is intended to load.

In addition, do not enter a value for Dataset File Name or the Server OS Filename in the Execution Setup definition. The person submitting the Load Set for execution must

enter a file name. The system uploads the file at the time that you specify it. If you specify it during Load Set or Execution Setup definition, the data in the file may be out of date by the time the user runs the Load Set.

> **Note:**
>
> It does not make sense to schedule a SAS Load Set for repeated execution because it would use the same BLOB ID each time, reloading the same data.
>
> If you schedule a single run of a SAS Load Set for a later time, it loads the data that was current at the time you specified the file.

Text

This section contains the following topics:

- About Text Load Sets
- UTF8 Encoding
- Text Load Set Attributes
- Text Load Set Table Descriptors
- Text Load Set Parameters
- Text Load Set Planned Outputs
- Text Load Set Execution Setup

About Text Load Sets

A Text Load Set loads data in a text file in either fixed or delimited format into one or more Table instances in Oracle LSH.

During Text Load Set definition, you can upload a metadata file. The system reads the data structure from the file and creates a Table Descriptor with the same structure. You can load multiple metadata files zipped together and create multiple Table Descriptors at the same time.

During Text Load Set execution, you can include metadata files in a zipped file with data files. For each metadata file, the system creates a matching Table Descriptors if none exists or updates and synchronizes a Table Descriptor if it does exist, and loads data from the data file(s). The system identifies files with extension .mdd as a metadata definition file and treats other extensions as data files.

You can specify tolerance factors to determine whether or not to continue loading if errors occur. See Text Load Set Parameters.

> **Note:**
>
> All text files must use carriage returns between records so that each record appears on its own line.

UTF8 Encoding

To ensure that Oracle LSH stores and displays special characters in your data correctly, convert text files that are not already using UTF8 encoding to UTF8 before uploading them to Oracle LSH.

To do this, copy the text in the text file into a a text editor such as Notepad or Wordpad and use the **Save As** option to select UTF-8 as the **Encoding** value.

Text Load Set Attributes

The only attribute is the **Save Input File?** flag:

- If set to **Yes**, the system saves the uploaded file as an output. This is especially useful when uploading data files with multiple date formats; see Text Load Set Planned Outputs.
- If set to **No**, and the file was uploaded from the user's desktop, the uploaded file is deleted after the load completes. This is the default value. If the file is uploaded from the operating system, it is not deleted.

Text Load Set Table Descriptors

Oracle LSH can read the data structure from a metadata (.mdd) file using a specified format (see Example 7-1) and define a Table Descriptor or read multiple metadata files zipped together to create multiple Table Descriptors. You can also define a target Table Descriptor manually; see Adding a New Target Table Descriptor .

You can create multiple Table Descriptors by uploading a zip file containing multiple metadata files. The system unzips the file, identifies and reads the metadata files (.mdd extension) and ignores any other files. It uses the metadata information to either create new Table Descriptors or update and synchronize existing Table Descriptors as the case may be.

- Creating a Metadata File
- Delimiter
- Table and Table Descriptor Attributes
- Columns

Creating a Metadata File

If the file has an .mdd extension, the system expects a set of Column attribute values, optionally preceded by a row identifying the delimiter and a row defining Table attribute values, each of which must begin with a key word. A row beginning with dashes is treated as a comment. For example:

```
--This is a comment.
```

Example 7-1 Text Load Set Metadata File

```
lsh_delimiter = ,
--This section is for the Table attributes
lsh_table= DM,DEMOG Table,EMP,EMP,EMP,Staging with
Audit,Yes,Yes,Blinded,target,yes,yes
```

```
--The following are columnsComment Line 2
INITS,VARCHAR2,100,,inits,inits,2.,,inits,no,1
AGE,NUMBER,10,,2,age,age,2.,,age,yes,1
    DOB,DATE,,,3,dob,dob,datetime.,,dob,yes,1,MM/DD/YY HH24:MI:SS
```

Delimiter

The first row defines the delimiter used in the file. If not specified, Oracle LSH treats it as a comma delimited file. The delimiter row must begin with `lsh_delimiter=`

Table and Table Descriptor Attributes

The second row lists the table attributes required in the file. The Table attribute row must begin with `lsh_table=`

If the second row is not present or contains null values, the system assumes that the filename (without extension) is the Table Name and follows the normal Oracle LSH default behavior for the attribute values. The attributes and their required order in the file are: `Name, Description, Oracle Name, SAS Name, SAS Label, Process Type, Allow Snapshot?, Blinding Flag?, Blinding Status, SAS Library Name, Is Target?, Target as Dataset?, SDTM Identifier`

Some attributes have associated reference codelists and allow either the actual values for the associated reference codelist (RC) columns or the decode values defined in the "Meaning" attribute of the _RC lookup. For example, "select meaning from cdr_lookups where lookup_code='< RC>' so that **YES** or **Yes** or **$YESNO$YES** are acceptable values.

The table below outlines the applicable values for each attribute that has an associated reference codelist.

> **Note:**
>
> Processing types that require audit keys are not supported.

Table 7-1 Table Attributes with Reference Codelist Values

Attribute	Values
Processing Type	Staging with Audit, Staging without Audit, Transactional High Throughput, Transactional without Audit
Allow Snapshot	Yes, No
Blinding Flag	Yes, No
Blinding Status	Blinded, Unblinded
Is Target	Yes, No
Target as dataset	Yes, No
Data Type	Date, Number, Varchar2
Nullable	Nullable Yes, No
SDTM Identifier	SUBJECT/SUBJECTVISIT

Columns

Subsequent rows must contain the column and variable attributes with each represented by a new row in the text file with attributes. The position is determined by the order in which the column /variable rows are processed. For example, in a comma delimited file: `Name, Data Type, Length, Precision, Oracle Name, SAS Name, SAS Format, Description, SAS Label, Nullable, Default Value, Date Format`

Normal Oracle LSH validation rules apply to the Column or variable attributes. The operation uses Oracle LSH default values if invalid values are provided for any of the attributes.

Table 7-2 Column Attributes with Reference Codelist Values

Attribute	Valid Values
Data Type	VARCHAR2, NUMBER, DATE
Nullable	YES, NO
SDTM Identifier	

Text Load Set Parameters

Text Load Sets include the Parameters listed below. You can change their default values in the Execution Setup definition. The user can change their values at runtime.

> **Note:**
>
> Because these are predefined, required Parameters you should change only the values specified below or the Load Set may not function properly. Do not change any other properties of these parameters.

- **Data File Name**. Do not select a file when defining the Load Set. Oracle LSH uploads the file when you submit the Load Set's Execution Setup. If you upload the file during Load Set definition, the data will probably be outdated at the time the Load Set is executed. Therefore, leave this field blank. The user must enter the filename at runtime if **Load From Server OS** is set to No.
- **Data Format**. Choose either Fixed or Delimited:
  - **Fixed**. The system uses the Load Set's target Table Descriptor Column definitions to interpret the data type and length of the values in the text file. The file must contain the correct number of characters for each value in each column of each record.

    In the case of Table Descriptor Columns with a Number data type (with a fixed format file), if you data includes a sign (+/-), you must increase the length by one to accommodate the positive (+) or negative (-) sign. In addition, if the data includes numbers with decimal places, you must declare the precision and increase the length by one to accommodate the decimal marker. See Defining Table Columns for further information.

  - **Delimited**. With delimited records, you specify the character used in the source file between each record as the separator character. Different records in the same column can be of different lengths, up to the maximum allowed. The system loads the contents between the separator character into each consecutive Column of the target Table instance.

    For example, if you have a Table instance with Columns Patient ID, Patient Initials, and Date of Birth, and the separator character was a comma, and the date format was DDMMYYYY, the first two records might look like this:

    ```
    54602,EKP,04081949

    66781,BAH,22011955
    ```

- **Delimiter Character**. (Required for Delimited-format Load Sets only.) Specify the character(s) used as the value delimiter in the source text file. In the example above, the comma (,) is the separator character. The default character is the comma (,). You can use any two unicode characters as a delimiter. For example, to use tab as the delimiter character, enter `\t`.

  > **Note:**
  >
  > Even if you specify that this is a fixed format Load Set, this Parameter is displayed. Leave the default value as is.

- **Enclosing Character**. (For Delimited-format Load Sets only; recommended but not required.) If any record value may contain the delimiter character, you need an enclosing character. Specify the character used to enclose each value. The default character is double quotation marks (").

  For example, if a double quotation mark (") were the enclosing character, the same two records would look like this in the source text file:

  ```
  "54602","EKP","04081949"

  "66781","BAH","22011955"
  ```

  You can use any single unicode character as an enclosing character.

  > **Note:**
  >
  > Even if you specify that this is a fixed format Load Set, this Parameter is displayed and is required. Leave the default value as is; the system does not use this value.

  > **Note:**
  >
  > Both delimiter and enclosing characters should be characters that never appear in the data content of the file.
  >
  > The delimiter character and the enclosing character must be different. If they are the same, the Load Set execution will fail.

> **Note:**
>
> If you use a tab as the delimiter and specify an enclosing character, the following limitations apply:
>
> – Tabs cannot be part of the data even if the data is enclosed.
>
> – The length of the data, including enclosing characters and duplicate enclosing characters, cannot exceed 4000 characters.

- **Initial Records to Skip**. If you want to not load records at the beginning of the file, enter the number of records you want the system to skip. The default value is zero (0).
- **Maximum Allowed Errors**. Tolerance factor; the maximum number of invalid rows you are willing to tolerate before the SQL*Loader stops the load process and marks the Load Set job as failed. The default value is 0.
- **Temp LOB ID**. This Parameter is for internal use only. Do not modify it.
- **Date Format**. Enter the date format used in the source data or .mdd file, if any. The date format you enter here must correspond exactly with the date field in the source text file, else the Load Set cannot execute correctly. Do not enter a value here if the source text file does not contain a date field.
- **Load From Server OS**. You have the option to load a file from your local computer or from a remote system. Set this parameter to Yes (default is No) if your file is either on the Oracle LSH DP server or on another computer that has its drives NFS-mounted on the DP server computer. If you set this parameter to No, you must upload a text file from your local computer when running the Text Load Set job by providing its name in the Data File Name parameter.

> **Note:**
>
> The system can load remote text files only from an Oracle LSH DP Server that is running the SQL*Loader service.

- **Server OS Filename**. If you set Load From Server OS to Yes, enter the absolute path of the remote file with the filename and extension, else leave this parameter blank.
- **Direct**. If set to Yes (default is No), the system uses the direct path INSERT to load data from the text file into the Oracle LSH target Table. If the Table Descriptor mapped to the Text Load Set is not of the Transactional High Throughput processing type, the system loads data using the conventional INSERT and writes warning messages to the job log.

> **Note:**
>
> See the Oracle Database Utilities Guide (part number B14215-01) for more information on this SQL*Loader Command-Line Parameter.

- **(Optional) Rows**. This parameter applies only when you set Direct to Yes.

It indicates the number of rows that SQL*Loader reads together at one time from the datafile before writing them to the target Table and committing to the database. If you leave this parameter blank, by default the system reads all the rows from the datafile and then writes the data to the target Table.

For conventional path loads, (if you set Direct to No), this parameter indicates the number of rows that SQL*Loader assigns to the bind array. If you leave this parameter blank, Oracle LSH calculates the number of rows.

> **Note:**
>
> A very high or a very low value for Rows can adversely affect the system's performance. Enter a value only if you are absolutely sure that you need to change this.
>
> See the Oracle Database Utilities Guide (part number B14215-01) for more information on this SQL*Loader Command-Line Parameter.

- **Drop and Recreate Indexes**. If set to Yes (default), and if the value of the parameter Direct is set to Yes, the system drops all non-unique indexes before running a data-loading job and recreates the non-unique indexes after loading data into the target Tables. The system does not drop indexes if Direct is set to No.
- **Recoverable**. This parameter applies only when you set Direct to Yes.

  If set to Yes (default), the system can recover data in the event of a failure. If set to No, the data becomes unrecoverable because SQL*Loader does not maintain a redo log file for the data.

> **Note:**
>
> For more information on the SQL*Loader UNRECOVERABLE clause at: *Oracle® Database Utilities* at `https://docs.oracle.com/cd/E11882_01/server.112/e22490/toc.htm`.

Text Load Set Planned Outputs

The system automatically creates four Planned Outputs for Text Load Sets:

- **Text Loading Log File (.log)**. The system writes the job log into the log file.
- **Text Loading Control File (.ctl)** The system generates the SQL*Loader control file and writes the SQL*Loader options to this file. This is useful when you use bulk loading.
- **Text Loading Error File (.err)**. The system writes each invalid record that got rejected while writing data into a target Table instance into this Planned Output.
- **Text Loading Input Data File**. If you set the **Save Input File** attribute to **Yes**, Oracle LSH uses a Planned Output as a placeholder for the input file saved as an output during execution. If you set **Save Input File** to **No**, the system still creates the Planned Output but does not create the actual output when you run the Load Set.

If you upload a zipped file with **Save as Input** set to **Yes**, the system always saves it as TEXTLOAD.ZIP—it does not have its original name when saved.

Setting **Save as Input** to **Yes** when you are loading multiple zipped files is especially useful when uploading data files with multiple date formats. If the date formats are the same, the **Save as Input** setting is immaterial. If the date formats vary, and the **Save as Input** option is set to **Yes**, the system uses the date format specified in each metadata file.

You can classify the Planned Output and view the actual output through the Oracle LSH Outputs user interface, subject to normal Oracle LSH security rules. To view the contents of the file, you must download it and view it through SAS.

Text Load Set Execution Setup

When you define an Execution Setup for a Text Load Set, be careful to give it a name and description that will enable a person submitting the Load Set for execution to understand which file should be loaded.

The file must contain data in a format that corresponds to the Table Descriptors defined for the Load Set, but the Table Descriptors are created as part of the Load Set definition, while the file can only be specified at runtime. Therefore, provide enough information in the Execution Setup Name and Description to make it clear which data this Load Set definition is intended to load.

In addition, do not enter a value for Data File Name or Server OS Filename in the Execution Setup definition. The system uploads the file at the time that you specify it. If you specify it during Load Set or Execution Setup definition, the data in the file may be out of date by the time the user runs the Load Set.

> **Note:**
>
> It does not make sense to schedule a Text Load Set for repeated execution because it would use the same BLOB ID each time, reloading the same data.
>
> If you schedule a single run of a Text Load Set for a later time, it loads the data that was current at the time you specified the file.

Oracle Clinical Stable Interface

This section contains the following topics:

- About Oracle Clinical Stable Interface Load Sets
- OC Stable Interface Tables Load Set Attributes
- OC Stable Interface Tables Load Set Table Descriptors
- OC Stable Interface Tables Load Set Parameter
- OC Stable Interface Tables Load Set Planned Outputs
- OC Stable Interface Tables Load Set Execution Setups

About Oracle Clinical Stable Interface Load Sets

Oracle Clinical Stable Interface Load Sets give you access to the metadata of all Oracle Clinical tables that are part of Oracle Clinical's stable interface.

For information on Oracle Clinical tables and joins, request a copy of the *Oracle Clinical Stable Interface Guide* from Oracle Support. This documentation is available only to Oracle Clinical customers.

OC Stable Interface Tables Load Set Attributes

Click **Update** and enter a value for the following attribute:

Remote Location. Click the Search icon and select a source remote location/ connection combination from the list of values.

OC Stable Interface Tables Load Set Table Descriptors

Oracle LSH automatically creates Table Descriptors from the Oracle Clinical tables you specify.

To specify the Oracle Clinical tables to load, do the following:

1. Click **Upload Table Descriptors**.
2. Select one or more Oracle Clinical views to load.

   Refer to the Oracle Clinical documentation for information on these tables. Functional information is in *Creating a Study* and *Conducting a Study*. Table structure information is in the *Oracle Clinical Stable Interface Guide*.
3. Click **Apply**. The system returns you to the Load Set screen and displays the Table Descriptors in the Table Descriptors subtab.
4. Map the Table Descriptors to Table instances.

OC Stable Interface Tables Load Set Parameter

Oracle Clinical Labs Load Sets have the following runtime Parameter:

Remote Location. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.

OC Stable Interface Tables Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Stable Interface Tables Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Stable Interface Tables Load Set Parameter.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

Oracle Clinical Data Extract SAS Views

This section contains the following topics:

- About OC DX SAS Views
- OC DX SAS Views Load Set Attributes
- OC DX SAS Views Load Set Parameters
- OC DX SAS Views Load Set Table Descriptors
- OC DX SAS Views Load Set Planned Outputs
- OC DX SAS Views Load Set Execution Setups

About OC DX SAS Views

Data Extract (DX) SAS Views Load Sets allow you to make Oracle Clinical patient data available in Oracle LSH using the SAS DX Views your company has already defined in Oracle Clinical. There is no need to redefine the views in Oracle LSH.

You can load views from one Study Access Account or Study Set Access Account at a time.

If you use SAS macro and format libraries to create your Oracle Clinical DX SAS Views, you must define a LSH Program to store them in LSH. LSH includes Program types especially for this purpose.

OC DX SAS Views Load Set Attributes

Click **Update** and enter values for the following attributes:

- **Remote Location**. Click the Search icon and select a source remote location/ connection combination from the list of values.
- **Study Name**. Click the Search icon and select the name of the Study or Study Set whose views you want. The system displays all studies that your connection has access to in Oracle Clinical.

OC DX SAS Views Load Set Parameters

Oracle Clinical Data Extract SAS Views Load Sets have the following runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must select this or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **View Type**. From the list of values, select the Oracle Clinical Study or Study Set Access Account that maintains the views you want.

> **Note:**
>
> Because this is a predefined, required Parameter you should not change anything except the default value in the Load Set definition, or the Load Set may not function properly.

OC DX SAS Views Load Set Table Descriptors

You can upload views from the Remote Location and study you specified. Oracle LSH uses the structure of the Oracle Clinical views to create the target Table Descriptor and its underlying Table definition.

When you install an OC SAS DX Load Set using the **Install** button on the Load Set screen, if the Load Set has no Table Descriptors, the system creates a target Table Descriptor for each active SAS DX View at the specified Remote Location for the specified Oracle Clinical study or study set. It also creates a matching Table instance in the current Work Area for each Table Descriptor and maps the matching Table Descriptor and Table instance.

OC DX SAS Views Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC DX SAS Views Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC DX SAS Views Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

Oracle Clinical Data Extract Views

This section contains the following topics:

- About OC Data Extract View Load Sets
- OC Data Extract Views Load Set Attributes
- OC Data Extract Views Load Set Table Descriptors
- OC Data Extract Views Load Set Parameters
- OC Data Extract Views Load Set Planned Outputs
- OC Data Extract Views Load Set Execution Setups

About OC Data Extract View Load Sets

Oracle Clinical Data Extract (DX) Views Load Sets allow you to make Oracle Clinical patient data available in Oracle LSH using the Oracle DX Views your company has already defined in Oracle Clinical. There is no need to redefine the views in Oracle LSH.

You can load views, including union views, from one Access Account at a time.

You can either physically load patient records into Oracle LSH or use the Load Set as a pass-through view to Oracle Clinical; see Oracle Tables and Views.

You can load views of any type—stable, snapshot, or current—that are maintained in the specified access account. However, you can see data in current Oracle Clinical DX Views in Oracle LSH only if you do not physically load the data into Oracle LSH.

There are two types of Oracle Clinical data extract view adapters: regular and incremental. The regular one uses full-mode reload processing. The incremental adapter uses incremental transactional processing, in either full or incremental load. For fastest performance, create the Load Set using the Incremental type and select incremental processing when you run the Load Set.

Table 7-3 OC Data Extract View Load Set Types

Load Set Type	OC DX View Types Allowed	Processing Type Used	Processing Options Available at Runtime	Deletion Behavior
Regular	Stable, Snapshot, or Current	Reload	Full	Soft-deletes records that are not reloaded. See Reload Processing.
Incremental	Stable	Transactional, with or without audit (set in a profile, can be overwritten for a target table instance)	Full or incremental	In **Full** mode, the adapter first deletes all data and then loads all current data: • If you are using Transactional with Audit processing, the original data is soft-deleted. The data remains in the database with an end timestamp reflecting the time of the job. • If you are using Transactional without Audit processing, the original data is hard-deleted. No record of the data is maintained. In **Incremental** mode, uses DML statements to insert, update, or soft-delete changed records. In addition, data that has been hard-deleted in Oracle Clinical is hard-deleted in Oracle LSH.

OC Data Extract Views Load Set Attributes

Click **Update** and enter values for the following attributes:

- **Remote Location**. Click the Search icon and choose a source database/ connection combination from the list of values.
- **Study Name**. From the list of values, select the name of the Study or Study Set whose views you want. The system displays all studies that your connection has access to in Oracle Clinical.
- **View Type**. From the list of values, select the type of view you want to load; for example, Stable, Snapshot, or Current.

OC Data Extract Views Load Set Table Descriptors

Use the **Upload Table Descriptors** function to specify the tables you want to load. The system generates a list of tables in the study in the location that you specified in the Attributes section.

When you install an OC DX Load Set using the **Install** button on the Load Set screen, if the Load Set has no Table Descriptors, the system creates a target Table Descriptor for each active Oracle DX View at the specified Remote Location for the specified Oracle Clinical study or study set. It also creates a matching Table instance in the current Work Area for each Table Descriptor and maps the matching Table Descriptor and Table instance.

OC Data Extract Views Load Set Parameters

Oracle Clinical Data Extract Oracle Views Load Sets have the following runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must enter his or her own remote location/connection combination (or a remote location and shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value.
- **View Type**. From the list of values, select the Oracle Clinical Study or Study Set Access Account that maintains the views you want.

> **Note:**
>
> Because this is a predefined, required Parameter you should not change anything except the default value in the Load Set definition, or the Load Set may not function properly.

OC Data Extract Views Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Data Extract Views Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Data Extract Views Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

Oracle Clinical Design and Definition

This section contains the following topics:

- About OC Design and Definition Load Sets
- OC Design and Definition Load Set Attributes

- OC Design and Definition Load Set Table Descriptors
- OC Design and Definition Load Set Parameters
- OC Design and Definition Load Set Planned Outputs
- OC Design and Definition Load Set Execution Setups

About OC Design and Definition Load Sets

Oracle Clinical Design and Definition Load Sets load the metadata on which CRFs are based, including:

- Data Collection Modules (DCMs)
- Data Collection Instruments (DCIs)
- Procedures
- Copy Groups
- Data Extract Queries and Templates

OC Design and Definition Load Set Attributes

Click **Update** and enter values for the following attributes:

- **Remote Location**. Click the Search icon and choose a source database/ connection combination from the list of values.
- **Design Sub-System**. Click the Search icon and select either Study Design or Study Definition. You can load tables from only one subsystem in a single Load Set. For a list of the tables you can load from each subsystem, see OC Design and Definition Load Set Table Descriptors.

OC Design and Definition Load Set Table Descriptors

Oracle LSH automatically creates Table Descriptors from the Oracle Clinical tables you specify. The tables available depend on the subsystem you specified.

To specify the Oracle Clinical tables to load, do the following:

1. Click **Upload Table Descriptors** to specify the Oracle Clinical tables you want to load. Oracle LSH generates a list of tables in the study in the location that you specified in the Attributes section.
2. Select one or more Oracle Clinical tables to load. The choices available depend on which subsystem you selected as an attribute.

   Refer to the Oracle Clinical documentation for information on these tables. Functional information is in *Creating a Study*. Table structure information is in the *Oracle Clinical Stable Interface Guide*.
3. Click **Apply**. The system returns you to the Load Set screen and displays the Table Descriptors in the Table Descriptors subtab.
4. Map the Table Descriptors to Table instances.

- Study Design Table Descriptors
- Study Definition Table Descriptors

Study Design Table Descriptors

If you set the Design Sub-System attribute to Study Design, you can load any of the following tables:

Blind Breaks
Clinical Planned Events
Clinical Planned Processes
Clinical Procedures
Clinical Studies
Clinical Study History
Clinical Study Objectives
Clinical Study States
Clinical Study Versions
Clinical Study Version Sizes
Clinical Subjects
Clin Study Enrollment Criteria
Clin St Termination Criteria
Combined Treatment Components
Daily Doses
Enrollment Plans
Factors
Interval Treat Regimen Assign
OCL Dosage Forms
OCL Investigators
OCL Organization Units
OCL Product Masters
OCL Programs

Study Definition Table Descriptors

If you set the Design Sub-System attribute to Study Definition, you can load any of the following tables:

Copy Groups
Copy Group Details
Correlation Items
Data Extract Views
DCIs
DCI Books
DCI Book DCI Constraints
DCI Book Pages
DCI Book Physical Pages
DCI Form Versions
DCI Modules
DCI Module Pages
DCMs
DCM Conditional Branches
DCM Layout ABS Pages
DCM Layout Graphics

DCM Layout Text
DCM Questions
DCM Question Groups
DCM Ques Repeat Defaults
DCM Schedules
Procedures
Procedure Details
Procedure Questions
Procedure Question Groups
Procedure Texts
Procedure Variables
Proc Det Var Usage
Queries
Query Details
Query Key Cols
Query Where
Query Where Cols
Templates
Template Columns
Template Indexes
Template Index Cols
Unions
View Question Mappings
View Restrictions
View Restriction Cols
View Template Questions

OC Design and Definition Load Set Parameters

Oracle Clinical Design and Definition Load Sets have the following runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **Study**. The user submitting the Load Set for execution must specify the study from which to load data.

> **Note:**
>
> Because this is a predefined, required Parameter you should not change anything except the default value in the Load Set definition, or the Load Set may not function properly.

OC Design and Definition Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Design and Definition Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Design and Definition Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

Oracle Clinical Global Metadata

This section contains the following topics:

- About OC Global Metadata Load Sets
- OC Global Metadata Load Set Attributes
- OC Global Metadata Load Set Table Descriptors
- OC Global Metadata Load Set Parameters
- OC Global Metadata Load Set Planned Outputs
- OC Global Metadata Load Set Execution Setups

About OC Global Metadata Load Sets

An Oracle Clinical Global Metadata Load Set loads your Oracle Clinical Global Library definitions and automatically converts them to Oracle LSH definitions as follows:

- Oracle LSH loads all Oracle Clinical Questions and converts them to Oracle LSH Variables.
- If a Question is associated with a Discrete Value Group (DVG) in Oracle Clinical, Oracle LSH converts the Question to a Parameter and converts its DVG values to a list of allowable values for the Parameter.
- Oracle LSH loads all Oracle Clinical Question Groups and converts them to Oracle LSH Table definitions with Columns based on the Variables corresponding to each Question in the Question Group.

The first time you run an Oracle Clinical Global Metadata Load Set, the system creates an Oracle LSH Domain called "Oracle Clinical Global Libraries" and an Application Area for the particular Oracle Clinical Global Library Domain. The first time you run a Load Set for a different OC Global Library Domain, Oracle LSH automatically creates another Application Area for the new OC Global Library Domain.

> **Note:**
>
> The use of the term "Domain" is confusing because Oracle LSH and Oracle Clinical use the term differently. The system creates an Oracle LSH Domain to contain all Oracle Clinical Global Library Domains. Within this Domain, Oracle LSH creates an Oracle LSH Application Area for each OC Global Library Domain.

Each subsequent time you run a Load Set for a Global Library Domain, Oracle LSH add any new definitions and modifies any definitions that have been modified in Oracle Clinical. If an Oracle Clinical Question has been removed from a Question Group in Oracle Clinical, Oracle LSH removes the corresponding Column from the corresponding Table. However, Oracle LSH does not delete or retire any Oracle LSH object definitions if their corresponding Oracle Clinical object is retired.

Oracle Clinical Global Metadata Load Sets never delete any metadata because the definitions may be in use in Oracle LSH. The processing is the same in either Full or Incremental mode.

OC Global Metadata Load Set Attributes

There are no attributes.

OC Global Metadata Load Set Table Descriptors

No Table Descriptors are required or allowed.

OC Global Metadata Load Set Parameters

Oracle Clinical Global Metadata Load Sets include the Parameters listed below.

- **Remote Location**. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **Library Domain Name**. From the list of values, select the name of the Oracle Clinical Global Library Domain that you want to load. Do not change any of the other Parameter settings.

OC Global Metadata Load Set Planned Outputs

The only Planned Output for an Oracle Clinical Global Metadata Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Global Metadata Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Global Metadata Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

Oracle Clinical Labs

This section contains the following topics:

- About OC Labs Load Sets
- OC Labs Load Set Attribute
- OC Labs Load Set Table Descriptors
- OC Labs Load Set Parameters
- OC Labs Load Set Planned Outputs
- OC Labs Load Set Execution Setups

About OC Labs Load Sets

Oracle Clinical Labs Load Sets load lab reference ranges and associated information from Oracle Clinical Labs-related tables (see OC Labs Load Set Table Descriptors for a complete list).

The Lab Assignment Criteria table is not included. You can load it separately if necessary, using the Oracle tables adapter.

Because Oracle Clinical Lab data tables are closely interrelated and are used in multiple studies, you must load all lab tables each time you run the Load Set. However, you can choose the Labs for which to load information.

Each execution of this type of Load Set retrieves the most recent creation or modification timestamp from the Lab tables in Oracle Clinical.

OC Labs Load Set Attribute

Click **Update** and enter a value for the following attribute:

Remote Location. Click the Search icon and choose a source database/connection combination from the list of values.

OC Labs Load Set Table Descriptors

Do not define any Table Descriptors. The system automatically creates Table Descriptors for Lab tables. The Oracle Clinical tables loaded are:

Labs
Lab Panels
Lab Panel Questions
Lab Range Subsets
Lab Test Question Units
Lab Units
Lab Unit Conversions
Preferred Lab Units
Preferred Lab Unit Groups
Ranges

> **Note:**
>
> Do not change any of these Table Descriptors. They belong to the OC Labs adapter and any changes to them will make the adapter invalid causing Load Sets of this type to stop working.

You must map the Table Descriptors to Table instances. See Defining Table Descriptors for instructions.

OC Labs Load Set Parameters

Oracle Clinical Labs Load Sets have the following runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **Lab**. The user submitting the Load Set for execution can specify the lab for which to load data.

> **Note:**
>
> Because this is a predefined, required Parameter you should not change anything except the default value in the Load Set definition, or the Load Set may not function properly.

OC Labs Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Labs Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Labs Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups

Oracle Clinical Randomization

This section contains the following topics:

- About OC Randomization Load Sets
- OC Randomization Load Set Attributes
- OC Randomization Load Set Table Descriptor
- OC Randomization Load Set Parameters

- OC Randomization Load Set Planned Outputs
- OC Randomization Load Set Execution Setups

About OC Randomization Load Sets

Oracle Clinical Randomization Load Sets load real or dummy treatment pattern information for Oracle Clinical studies. Randomization Load Sets load data from a single Oracle Clinical table: TREAT_ASSIGN_ALL_VIEW.

This table contains two separate sets of data. One set contains the actual treatment codes that reveal which patient is receiving which treatments. This information is normally blinded in Oracle Clinical, and cannot be loaded into Oracle LSH until it has been unblinded in Oracle Clinical. The second set of data is dummy data that Oracle Clinical generates randomly.

Oracle LSH automatically sets the Blinding Flag of the target Table instance to **Yes**.

The same Load Set instance can load either the real treatment codes or the dummy data. To load dummy data, no special security privileges are required. To load the real codes, the user running the Load Set must satisfy security requirements in both Oracle LSH and Oracle Clinical:

- Oracle Clinical Blinding Security
- Blinding Security

Oracle Clinical Blinding Security

Security access to the real treatment codes within Oracle Clinical is controlled by the Randomization Access Status Code (RAND_ACC_STAT_TYPE_CODE). The Oracle Clinical Randomization Adapter checks the value of this code in Oracle Clinical. The value in Oracle Clinical determines whether the real data can be loaded or only the dummy data, and determines the value set for the Blinding Status of the target Table instance:

- If the value is either Open or Release, the Load Set can load the real codes. The system sets the Blinding Status of the target Table instance to **Unblinded**.
- If the value is Access, and the RXA_ACCESS.TREAT_ACCESS_*STUDY* view exists in the user account being used to connect to Oracle Clinical, then the Load Set can load the real codes. The system sets the Blinding Status of the target Table instance to **Blinded**.
- If the user is trying to download the real codes and the RAND_ACC_STAT_TYPE_CODE value is anything other than Open, Release, or Access, the system raises an error and the load fails.

Blinding Security

To run a Randomization Load Set on dummy data, an LSH user must have normal security access to the Load Set instance and to the target Table instance.

In addition, to load the real treatment codes, the user must also have special blinding-related privileges on the target Table instance within the same User Group through which he or she has security access to the Table instance:

- If the data is currently blinded, the user must have Blind Break privileges.

- If the data has been unblinded, the user must have Read Unblind or Blind Break privileges.

If the user has Unblind privileges, he or she can permanently unblind the data within LSH; see Unblinding Table Instances. Even after data has been permanently unblinded, users must have Read Unblind privileges to view the data.

OC Randomization Load Set Attributes

Oracle Clinical Randomization Load Sets have no attributes.

OC Randomization Load Set Table Descriptor

The system automatically creates a Table Descriptor (and its underlying Table definition) for the Oracle Clinical table TREAT_ASSIGN_ALL_VIEW.

> **Note:**
>
> Do not change this Table Descriptor. It belongs to the OC Randomization adapter and any changes to it will make the adapter invalid.

You must map the Table Descriptors to Table instances. See Defining Table Descriptors for instructions.

OC Randomization Load Set Parameters

Oracle Clinical Labs Load Sets have the following runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **Study**. The user submitting the Load Set for execution must specify the study for which to load real or dummy treatment codes.
- **Treatment Data Type**. If set to Dummy (the default value) the system loads the dummy data. If set to Real, the system loads the real treatment codes, if all security requirements are met.

OC Randomization Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Randomization Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Randomization Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups.

Oracle Clinical Study Data

This section contains the following topics:

- About OC Study Data Load sets
- OC Study Data Load Set Attributes
- OC Study Data Load Set Table Descriptors
- OC Study Data Load Set Parameters
- OC Study Data Load Set Planned Outputs
- OC Randomization Load Set Execution Setups

About OC Study Data Load sets

Oracle Clinical Study Data Load Sets load study-specific non-patient data into LSH, including:

- Discrepancies
- Data Clarification Forms (DCFs)
- Page tracking information
- Patient status information

See OC Study Data Load Set Table Descriptors for a complete list of tables that this Load Set type can load.

Each execution of this type of Load Set retrieves the most recent data from the selected tables in Oracle Clinical.

OC Study Data Load Set Attributes

Click **Update** and enter a value for the following attribute:

Remote Location. Click the Search icon and choose a source remote location/ connection combination from the list of values.

OC Study Data Load Set Table Descriptors

LSH automatically creates Table Descriptors and their underlying Table definitions from the Oracle Clinical tables you specify. LSH stores the Table definitions in the current Application Area.

To specify the Oracle Clinical tables to load, do the following:

1. Click **Upload Table Descriptors** to specify the Oracle Clinical tables you want to load. The system generates a list of tables in the study in the location that you specified in the Attributes section.
2. Select one or more Oracle Clinical tables to load. The choices are:

   Data Clarification Forms
   DCF Discrepancies
   DCF Discrepancies Hist
   DCF Pages

DCF Page Entries
DCF Print Status
DCF Status Tracking
Discrepancy Entries
Discrepancy Entries T
Discrepancy Entry Review Hist
Discrepancy Entry Review Hist T
Patient Positions
Patient Positions T
Patient Positions History
Patient Statuses
Received Pages
Received Pages T
Received Page History
Received Page History T
Validation Reported Values
Validation Reported Values T

Refer to the Oracle Clinical documentation for information on these tables. Functional information about Data Clarification Forms (DCFs), discrepancies, page tracking, and validated reported values is in *Conducting a Study*. Functional information on patient positions and statuses is in *Creating a Study*. Table structure information on all tables is in the *Oracle Clinical Stable Interface Guide*.

3. Click **Apply**. The system returns you to the Load Set screen and displays the Table Descriptors in the Table Descriptors subtab.
4. Map the Table Descriptors to Table instances.

OC Study Data Load Set Parameters

Oracle Clinical Study Data Load Sets have the following runtime Parameters:

- **Remote Location**. The user submitting the Load Set for execution must select his or her own remote location/connection combination (or a remote location with a shared connection) at runtime to ensure the proper security for the external database. Do not enter a default value or change any of the other Parameter settings.
- **Study**. The user submitting the Load Set for execution must specify the study from which to load data.

> **Note:**
>
> Because this is a predefined, required Parameter you should not change anything except the default value in the Load Set definition, or the Load Set may not function properly.

OC Study Data Load Set Planned Outputs

The only Planned Output for any Oracle Clinical Load Set is a log file. See About Load Set Planned Outputs for further information.

OC Randomization Load Set Execution Setups

Do not set the Remote Location Parameter in the Execution Setup. See OC Study Data Load Set Parameters.

For general information on defining Execution Setups, see Creating, Modifying, and Submitting Execution Setups.

Installing Load Set Instances

You can install a Load Set instance directly from its Properties screen, using the Install button, or in its Work Area (see Installing a Work Area and Its Objects).

When you install a Load Set instance using the **Install** button on its Properties screen:

- The system checks in the Load Set instance and definition, and also the Table instances in the current Work Area to which the instance is mapped.
- The system checks if the Load Set is installable. If not, the system performs Automatic Mapping by Name on any unmapped Table Descriptors. If the Load Set is still not installable and there are still unmapped Table Descriptors, the system creates Table instances in the current Work Area from the Table Descriptors and maps them.

  In the case of Oracle Clinical DX Load Sets and Oracle Clinical SAS DX Load Sets, if the Load Set has no Table Descriptors, the system creates a target Table Descriptor for each active Data Extract View at the specified Remote Location for the specified Oracle Clinical study or study set. It also creates a matching Table instance in the current Work Area for each Table Descriptor and maps the matching Table Descriptor and Table instance.
- The system attempts to install the Load Set instance and the Table instances to which it is mapped. The system displays a success or error message. If the installation fails, the error message displays the name of any objects that were not installable.

> **Note:**
>
> If any of the mapped Table instances or the Load Set definition is not installable, the system cannot install the Load Set instance. See Installation Requirements for Each Object Type for the reasons these objects may not be installable.

> **Note:**
>
> To continue working on the Load Set, check it out.

- Access the Log File

Access the Log File

To see the log file for the installation, you must go to the Work Area Installation screen, as follows:

1. Click the **Applications** tab. The main Application Development screen opens.
2. Click the name of the Work Area you are working in. The Work Area screen opens.
3. From the **Actions** drop-down list, select **Installation History**.
4. Click **Go**. The system displays the Installation History screen with the log files in chronological order.
5. Click the **View Log** link for the most recent installation attempt or for the date and time that you ran the install process. The system displays the log file.

For information on installation and on reading the log file, see Installing a Work Area and Its Objects.

Modifying Load Sets

If you have the necessary privileges, you can modify a Load Set either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area. In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Load Set Instance Properties
- Modifying Load Set Definition Properties

Modifying Load Set Instance Properties

On the Load Set instance's Properties screen, click **Update** to enter changes. Oracle LSH creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must reinstall the Load Set for the changes to take effect.

You can modify the following:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field applies to the instance only. It specifies the Load Set definition to which this Load Set instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Table Descriptors, Parameters, and Planned Outputs.
  - Any new Table Descriptors are not mapped.
  - The Load Set's status changes to **Non Installable**.

  If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.

Modifying Load Set Definition Properties

You can go to a Load Set definition's Properties screen in one of the following ways:

- **From the Load Set's Properties screen**: Click the hyperlink of the Load Set definition that appears in the Definition field.
- **From the Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Load Set definition screen, click **Update** to enter changes. Oracle LSH creates a new version of the definition. You can change the following properties:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.

  You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.
- Modifying Table Descriptors
- Modifying Attributes and Parameters

- Modifying Planned Outputs

Modifying Table Descriptors

You cannot modify Load Set Table Descriptors because they must be identical to the external table or data set on which they are based. However, for most Load Set types you can add or remove Table Descriptors.

You can change the Table Descriptor mappings, which are part of the Load Set instance, not the definition. You do not need to check out the definition to modify the mappings. This may be useful if you want to load data into a standard Oracle LSH Table instance whose name or Column names differ from the source Table or data set.

Table Descriptors belong to the Load Set definition.

Modifying Attributes and Parameters

You cannot add or remove Attributes or Parameters because they are predefined for each Load Set type.

You can change some Parameter values in one or more Execution Setups. Select **Execution Setups** from the **Actions** drop-down list in the Load Set instance in the Work Area. See Creating, Modifying, and Submitting Execution Setups.

Modifying Planned Outputs

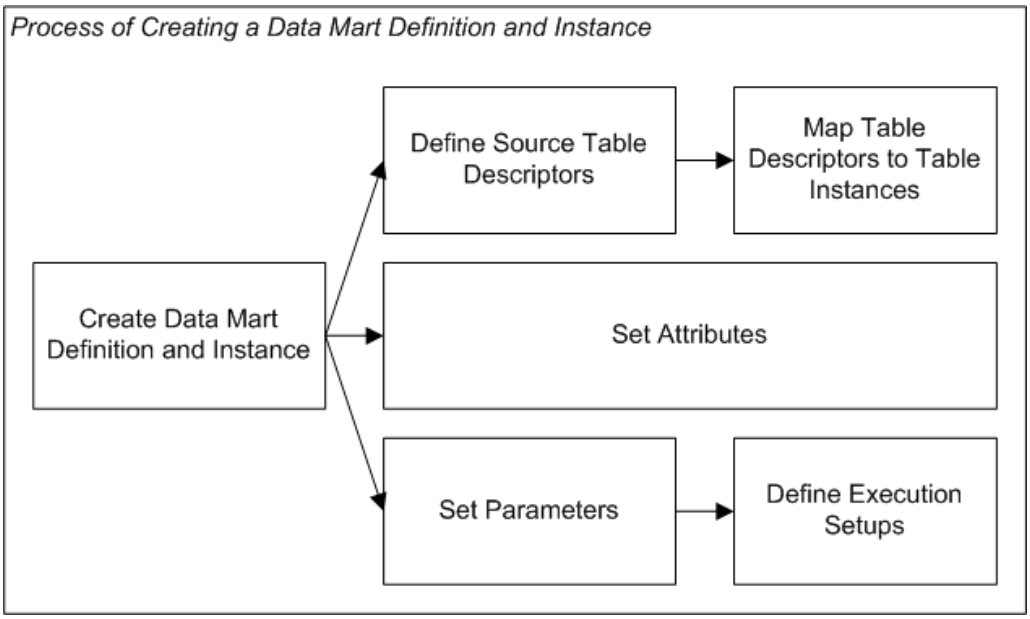
You cannot add or remove Planned Outputs because they are predefined for each Load Set type.

You can change the Planned Outputs' classifications, which affect the classifications of the actual outputs. See Classifying Outputs for further information.

Planned Outputs belong to the Load Set definition.

8
Defining Data Marts

Figure 8-1 Process of Creating a Data Mart Definition and Instance



This section contains information on the following topics:

- About Data Marts
- Creating a Data Mart
- Using the Data Mart Properties Screen
  If you are working in a Work Area, you see the properties of both the Data Mart instance and the Data Mart definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.
- Defining Table Descriptors
- Setting Data Mart Parameter Values
- About Data Mart Planned Outputs
- Defining Different Types of Data Marts
  Oracle LSH automatically creates Attributes, Parameters, and Planned Outputs for Data Marts based on the Data Mart type.
- Installing Data Mart Instances
- Modifying Data Marts

About Data Marts

A Data Mart exports all the data in a set of Oracle Life Sciences Data Hub (Oracle LSH) Table instances to one or more files for the purpose of recreating Oracle LSH data in an external system in a verifiable and reproducible manner. A Data Mart is an Oracle LSH primary executable object whose data file output is also called a Data Mart.

You can use Data Marts for many purposes, including:

- submitting data to a regulatory agency
- exporting a set of Oracle database tables to another system for data mining
- sharing data with a partner organization
- long-term data storage in text (or other) format

To include data from a Table instance in a Data Mart output, you create a Table Descriptor and map it to the Table instance. You can include data from any number of Table instances in a single Data Mart. As with other executables you can run a Data Mart on current data or on a snapshot, depending on settings in its Execution Setup.

Data Mart Types: You can create Data Marts in the following formats:

- Text Export
  - Text Fixed Format
  - Text Delimited Format
- SAS Export
  - SAS Data Sets
  - SAS Transport (CPORT or XPORT)
- Oracle Export

Execution: You define Execution Setups for Data Marts the same way you do for other Oracle LSH executables. See Creating, Modifying, and Submitting Execution Setups for further information.

Reports on Data Mart Definitions and Instances: From the **Actions** drop-down list, you can generate reports that provide information on a Data Mart definition or instance; see System Reports for information.

Creating a Data Mart

When you create a Data Mart in a Work Area, you are actually creating an instance of a Data Mart definition.

To create a new Data Mart instance:

1. In a Work Area, select **Data Mart** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Data Mart screen.
3. Choose one of the following options:

- **Create a new Data Mart definition and instance**. Choose this option if no Data Mart definition exists that can meet your needs, either as it is or with some modification.
- **Create an instance from an existing Data Mart definition**. Choose this option if a Data Mart definition already exists that meets your needs.

  If you can adapt an existing Data Mart definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.

4. Depending on your choice, follow one of the following sets of instructions:

- Creating a New Data Mart Definition and Instance
- Creating an Instance of an Existing Data Mart

Creating a New Data Mart Definition and Instance

When you select **Create a new Data Mart definition and instance** in the Create Data Mart screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Description**. See Creating and Using Object Descriptions.
   - **Data Mart Type**. The options are Oracle Export, SAS Export, and Text Export. See Defining Different Types of Data Marts for further information.
2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Data Mart.

   The system opens the Properties screen for the new Data Mart instance.
4. See the following instructions:
   - Defining Table Descriptors
   - Setting Data Mart Attribute Values
   - Setting Data Mart Parameter Values
   - About Data Mart Planned Outputs

Creating an Instance of an Existing Data Mart

If you create an instance of an existing Data Mart, its Table Descriptors and Attribute and Parameter settings are already defined. See Creating an Instance of an Existing Definition for instructions.

After you have created the Data Mart instance, you must map the Table Descriptors to Table instances; see Mapping Table Descriptors to Table Instances for instructions.

Using the Data Mart Properties Screen

If you are working in a Work Area, you see the properties of both the Data Mart instance and the Data Mart definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See also Figure 8-1.

See Modifying Data Marts for information on modifying Data Marts.

This section contains the following topics:

- Instance Properties
- Definition Properties
- Data Mart Attributes
- Buttons

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field specifies the Data Mart definition to which this Data Mart instance points.

  You can upgrade to a new version of the same definition. See Upgrading to a Different Definition Version from an Instance.
- **Validation Status:** This field displays the current validation status of the Data Mart instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Data Mart: Installable or Non Installable. See Installation Requirements for Each Object Type.
- **Version:** This field displays the current version number of the Data Mart instance.
- **Version Label:** This field displays the version label, if any, for the current Data Mart instance version.

  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.

Definition Properties

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to modify Table Descriptors, Parameters, or Planned Outputs. However, you can change Table Descriptor mappings without checking out the definition. See Understanding Object Versions and Checkin/Checkout for further information.

- **Latest Version:** If set to **Yes**, this Data Mart instance is pointing to the latest version of the Data Mart definition. If set to **No**, this Data Mart instance is pointing to an older version of the Data Mart definition.
- **Checked Out By:** This field displays the name of the person who has the Data Mart definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Data Mart Type:** This field displays this Data Mart definition's type: Text Export, SAS Export, or Oracle Export. See Defining Different Types of Data Marts.
- **Validation Status:** This field displays the current validation status of the Data Mart definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Data Mart in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Data Mart: Installable or Non Installable. See Installation Requirements for Each Object Type.

Data Mart Attributes

- **Export File Name:** This attribute applies to all Data Mart types. It is the name of the file where the output of the Data Mart gets stored. This is the only attribute for SAS and Oracle Export types of Data Marts. See SAS Data Mart Attribute, and Oracle Export Data Mart Attribute. Also see Text Data Mart Attributes.
- **Mode:** This attribute applies only to Text Data Marts. It is the mode used to create the text export file: **Fixed** or **Delimited**. See About Text Data Marts for an explanation of the two modes.
- **File Name Extension:** This attribute applies only to Text Data Marts. A Text Data Mart output may be of **.txt** (Fixed mode) or **.csv** (Delimited mode) type. See Text Data Mart Attributes.
- Setting Data Mart Attribute Values

Setting Data Mart Attribute Values

You cannot add or remove attributes or Parameters. They are predefined for each type of Data Mart. You can change their values in the Data Mart definition, but not in the Execution Setup.

To change attribute values:

1. Click the **Update** button in the Data Mart Attributes section of the screen.
2. Modify attribute values as necessary. Most attributes have an allowed list of values. Use the Search icon to select one.
3. Click **Apply**.

For information on the attributes and Parameters specific to each type of Data Mart, see:

- Defining Text Data Marts

- Defining SAS Data Marts
- Defining Oracle Export Data Marts

Buttons

From a Data Mart instance in a Work Area, you can use the following buttons:

- **Install:** Click **Install** to install the Data Mart instance, including any mapped source Table instances in the same Work Area; see Installing Data Mart Instances. For a list of reasons a Data Mart instance may not be installable, see Installation Requirements for Each Object Type.
- **Submit:** Click **Submit** to run the Data Mart instance. Before you can run the Data Mart, you must install it and create an Execution Setup for it (select **Execution Setups** from the **Actions** drop-down list).
- **Update:** Click **Update** to modify the Data Mart instance properties. See Modifying Data Mart Instance Properties. You can also edit Data Mart Attributes by clicking the **Update** button in the Data Mart Attributes section of the screen. See Setting Data Mart Attribute Values.
- **Check In/Out and Uncheck:** Click these buttons to check out, check in, or uncheck the Data Mart definition. Different buttons are displayed in the Data Mart Definition Properties section depending on the Checked Out Status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of the person who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.

Defining Table Descriptors

For all Data Mart types, you specify the data to include in the output file by adding one or more source Table Descriptors to the Data Mart and mapping each one to a Table instance whose data you want to export to the file. Depending on the type of Data Mart, the system may create one file per Table Descriptor or one file for the Data Mart as a whole; see Defining Different Types of Data Marts.

Data Marts can have only source Table Descriptors. They write to files, not tables.

Oracle LSH automatically creates the Planned Outputs for a Data Mart based on the Data Mart type and on the source Table Descriptors you specify. If you do not want to include all the Columns in a source Table instance in the Data Mart output, delete the unnecessary Columns from the Table Descriptor mapped to that Table instance.

For instructions, see About Table Descriptors and Mapping Table Descriptors to Table Instances .

Setting Data Mart Parameter Values

You cannot add or remove Parameters. They are predefined for each type of Data Mart. You can change their value and other settings either in the Data Mart definition, in the Execution Setup definition, or allow the user to set their value in the Execution Setup at runtime.

To change Parameter values and other settings in the Data Mart definition:

1. Check out the Data Mart definition if it is not already checked out.
2. In the Parameters subtab, click the hyperlink in the **Name** column of the Parameter you want to modify.
3. Click **Update**.
4. Modify the following attribute values if necessary:

> **Note:**
>
> Because these are predefined and required Parameters you should not modify any attributes other than the following or the Data Mart may not function properly.

   - **Read Only**. To prevent users from changing the value of this Parameter at runtime, check Read Only. To allow users to change the value at runtime, leave unchecked.
   - **Prompt**. The prompt is the label displayed for the Parameter in the Execution Setup. You can change it if you want to.
   - **Default Value**. You can change the default value in the Parameters List of Values. If you also make the Parameter Read Only, the default value becomes the value used at runtime.
5. Click **Apply**.

For information on the attributes and Parameters specific to each type of Data Mart, see:

- Text Data Mart Parameters
- SAS Data Mart Parameters
- Oracle Export Data Mart Parameters

About Data Mart Planned Outputs

When you first create a Data Mart definition, the system creates one Planned Output for the .log file and one or more others, depending on the Data Mart type. The Filename attribute value determines the output file names. The system adds the extension appropriate for the Data Mart type.

In addition, for SAS Dataset mode and Text Data Marts, when you add a Table Descriptor to the Data Mart, the system creates a Planned Output whose file name is the Table Descriptor name (in lowercase) plus the extension appropriate for the mode:.sas7bdat or, for Text Data Marts, .txt or .csv.

You can modify the classification of a Planned Output, which affects the classification of the corresponding actual Data Mart output; see Classifying Outputs.

At runtime the system creates only the outputs appropriate for the mode and other settings. See:

- Text Data Mart Planned Outputs
- SAS Data Mart Planned Outputs

- Oracle Export Data Mart Planned Outputs

Defining Different Types of Data Marts

Oracle LSH automatically creates Attributes, Parameters, and Planned Outputs for Data Marts based on the Data Mart type.

This section includes information on the Parameters and Planned Outputs for each Data Mart type:

- Defining Text Data Marts
- Defining SAS Data Marts
- Defining Oracle Export Data Marts

Defining Text Data Marts

This section includes the following topics:

- About Text Data Marts
- Text Data Mart Attributes
- Text Data Mart Parameters
- Delimited-Format Text Data Mart Parameters
- Separator Character
- Enclosing Character
- Use Enclosing Character?
- Text Data Mart Planned Outputs

About Text Data Marts

There are two modes of Text-type Data Marts: Fixed and Delimited. Both modes produce one file for each Table Descriptor, named *Data_Mart_name*.txt or .csv. You can combine all the files in a single .zip file.

Fixed-format Data Marts use more space than delimited-format Data Marts but result in faster performance.

- Fixed Format Text Data Marts
- Delimited Format Text Data Marts

Fixed Format Text Data Marts

For a fixed-format Data Mart, Oracle LSH automatically uses the length defined for each Column of the Table Descriptor to determine the number of characters to dedicate to each column value in the output file.

- For Columns of data type Varchar2, the number of characters dedicated to a column in the output file equals the defined Column length.
- For Columns of data type Number, the system uses the Column length plus one character for the plus (+) or minus (-) sign and, if the Column's number is defined

as having precision, an additional character for the decimal point (or other decimal marker).

- For Columns of data type Date, the system uses the date format string that is part of the Column definition or, if no format string is defined, uses 20 characters to accommodate the default format, DD-MON-YYYY/HH24:MI:SS.

When the system generates the file, it inserts the actual value for each row and then adds the number of spaces necessary (if any) to reach the number of characters dedicated to that Column. At the end of each record, the system inserts the carriage return appropriate for the operating system.

Delimited Format Text Data Marts

For a delimited-format Data Mart, Oracle LSH inserts a character as a delimiter after every column value for every record (except the last) as well as a carriage return at the end of each record. You must specify the character to be used as the delimiter. In addition, you can specify a different character to be used as an enclosing character around each character and date column value, and specify whether or not the system should actually use the enclosing character; see Text Data Mart Parameters for further information.

Text Data Mart Attributes

Text Data Marts have the following predefined attributes. To change their values, click **Update** in the Attribute section of the screen.

- **Export File Name:** Enter the name you want to give the zipped file that holds the individual text files. The default value is `text_dm.zip`. You must enter lowercase text. If you remove the .zip as part of the value, the system adds it.

> **Note:**
>
> The system produces this zipped file only if at runtime the value of the Zip Result Flag Parameter is either Zip or Both.

- **Mode:** Click the Search icon and then select either **Fixed** or **Delimited**. See About Text Data Marts for an explanation of the two modes. The default value is Delimited.
- **File Name Extension:** Click the Search icon then select an extension for the file that will contain the actual data. Select **.txt** for fixed mode Data Marts and **.csv** for delimited mode.

Text Data Mart Parameters

Oracle LSH automatically creates Parameters for Text-type Data Marts. For information on how you can modify them, see Setting Data Mart Parameter Values.

> **Note:**
>
> Because these are predefined, required Parameters you should change only the default value, Read Only setting, or Prompt for each Parameter or the Data Mart may not function properly; see Setting Data Mart Parameter Values.

- **Zip Result Flag:** Select one of the following values as the default:
  - **No**. If set to **No**, the system does not generate a zipped file. The Data Mart output consists of one text file for each Table Descriptor and the log file. The system disregards the defined Export Filename.
  - **Zip**. If set to **Zip**, the system generates a single zipped file that includes the text files generated for each Table Descriptor and the log file.
  - **Both**. If set to **Both**, the system generates a text file for each Table Descriptor, creates a zipped file that contains all the text files and the log file, and also leaves a copy of each text file and the log file outside the zipped file.

  The default value is **Zip**.
- **FirstRow Desc:** If you are creating a fixed-format Text Data Mart, you must enter a value for this Parameter, which determines the way the Data Mart handles the first row for each Table Descriptor.
  - **Yes**. If set to **Yes**, the system inserts the Column names as the first row in the text file for each Table Descriptor. If the Column name is too long for its length, the system truncates it. If it is a different format (for example, a Column name in text for a data type Number Column), the system inserts a warning in the Data Mart log file.
  - **No**. If set to **No**, the system starts the file with the data of the first record, not the Column names.

  The system ignores this value for delimited text Data Marts.
- **Operating System:** Select your operating system: **UNIX** or **MS Windows**. The system inserts the appropriate carriage return for the operating system at the end of every record. If you are using Linux, select UNIX. The default value is UNIX.

Delimited-Format Text Data Mart Parameters

If you are creating a Delimited-format Text Data Mart, the following Parameters must each have a value. The system ignores these values for fixed-format text Data Marts. See Example 8-2 and Example 8-1 for further information.

Example 8-1 Delimited Text Export Data Mart with Separator and Enclosed Characters

If a single quotation mark (') were the enclosing character, the same two records would look like this:

```
'54602','EKP','04081949'

'66781','BAH','22011955'
```

Example 8-2 Delimited Text Export Data Mart with Separator Character Only

If your Table Descriptor has Columns Patient ID, Patient Initials, and Date of Birth, the separator character was a comma, and the date format was DDMMYYYY, the first two records might look like this:

```
54602,EKP,04081949

66781,BAH,22011955
```

Separator Character

The system appends the separator character (delimiter) to every column value of every row (except the last value, where there is a carriage return instead) in order to clarify where one value ends and the next one begins.

The separator character should not appear as part of any column value of any record. The default value is a comma (,). You can enter any two characters. For example, to specify tab as the separator character, enter `\t`.

Enclosing Character

The system prepends and appends the enclosing character to every character and date column value of every record. The character should be one that does not appear as part of any column value of any record. The default value is a double quotation mark (").

> **Note:**
>
> The delimiter character and the enclosing character must be different. If they are the same, the Data Mart execution will fail.

Use Enclosing Character?

Select **Yes** or **No**.

- **Yes**. If set to **Yes**, the system encloses every column value with the character you specify in the Enclosing Character Parameter.
- **No**. If set to **No**, the system does not use the enclosing character.

Text Data Mart Planned Outputs

Oracle LSH creates one Planned Output for each Table Descriptor in the Data Mart definition. At runtime, the system generates one .txt or .csv file for each Planned Output. The file type is determined by the value of the File Name Extension attribute.

Depending on the value of the Zip Result Flag Parameter, the system may also create a Planned Output for a zipped file to contain all the files. The file name for this output is *filename_attribute_value*.zip.

The system also creates a Planned Output for the log file, named *text_dm*.log.

You can classify a Planned Output in order to classify the actual output when it is generated; see Classifying Outputs for further information.

Defining SAS Data Marts

This section includes the following topics:

- About SAS Data Marts
- SAS Data Mart Attribute
- SAS Data Mart Parameters
- SAS Data Mart Planned Outputs

About SAS Data Marts

Oracle LSH supports the following types, or modes, of SAS Data Marts, when you have purchased and installed SAS with Oracle LSH:

- **SAS CPORT:** A SAS CPORT Data Mart consists of a single .cport file containing data from all the Table instances mapped to the Data Mart's Table Descriptors.
- **SAS XPORT:** A SAS XPORT Data Mart also consists of a single .xpt file containing data from all the Table instances mapped to the Data Mart's Table Descriptors.

> **Note:**
>
> For SAS XPORT format's compatibility with SAS v6, Oracle LSH truncates long Table Column names (variables in a SAS data set) to 8 characters. Also, because of a limitation in SAS v6, you must ensure that Table instances mapped to a SAS XPORT Data Mart do not contain character data longer than 200 characters, to prevent the Data Mart's execution from failing.

- **SAS Data Sets:** A SAS Dataset Data Mart includes one .sas7bdat file for each Table Descriptor. If you set the **Zip Results** Parameter to **Zip**, all the .sas7bdat files and the .log file are included in a single .zip file.

SAS Data Mart Attribute

SAS Data Marts have one attribute: **Filename**. Enter a descriptive name. The system gives this name to the Planned Output and to the actual Data Mart output file. Do not add a file extension.

> **Note:**
>
> This attribute must have a value. The Data Mart is not installable without a file name.

SAS Data Mart Parameters

Oracle LSH automatically creates Parameters for SAS Data Marts. You can change the values of the following required Parameters only in the Data Mart definition:

- **Mode:** Select the mode that corresponds to the type of output you need: CPORT, XPORT, or Dataset. The default value is CPORT.
- **Zip Results:** The Zip Results Parameter setting has an effect only for SAS Data Marts of the Dataset mode. CPORT and XPORT Data Marts cannot be zipped.

  For SAS Dataset Data Marts, select one of the following values:

  – **No**. If set to **No**, the system does not generate a zipped file. The Data Mart output consists of one sas7bdat file for each Table Descriptor, plus the log file.

  – **Zip**. If set to **Zip**, the system generates a zipped file that includes the sas7bdat file generated for each Table Descriptor and the log file. The system also generates and uploads the individual SAS data sets for each Table Descriptor as separate outputs.

  The default value is **Zip**.

SAS Data Mart Planned Outputs

Oracle LSH includes a predefined Planned Output for the .log file and for each SAS Data Mart mode. At runtime the system creates only the outputs appropriate for the mode and, in the case of Dataset Data Marts, the Zip Results Parameter. The names and file names for these Planned Outputs are:

- **CPORT**: SAS DM.cport (name), *filename_attribute_value*.cport (file name)
- **XPORT**: SAS DM.xpt (name), *filename_attribute_value*.xpt (file name)
- **Zipped Dataset**. SAS DM.zip (name), *filename_attribute_value*.zip (file name)
- **Dataset** (zipped or not): Both the name and filename are *table_descriptor_name*.sas7bdat.

All file names are in lowercase.

You can classify a Planned Output in order to classify the actual output when it is generated; see Classifying Outputs for further information.

Defining Oracle Export Data Marts

This section includes the following topics:

- About Oracle Export Data Marts
- Oracle Export Data Mart Attribute
- Oracle Export Data Mart Parameters
- Oracle Export Data Mart Planned Outputs

About Oracle Export Data Marts

Oracle LSH generates a command file for the Oracle Export utility to produce (in Table mode) a single Oracle Export file containing the data and metadata of all the Table

instances mapped to the Table Descriptors of the Data Mart. The names of the export tables are the same as their corresponding Table Descriptors.

You can send Oracle Export Data Marts only to another Oracle database. To import its data, the external Oracle database must use the Oracle Import utility.

The Oracle Export file created by the Data Mart holds the tables in alphabetical order.

See *Oracle® Database Utilities 11g Release 2 (11.2)*, at `https://docs.oracle.com/cd/E11882_01/server.112/e22490/toc.htm`.

Oracle Export Data Mart Attribute

There is only one attribute: the file name of the actual Data Mart output. The default name is: oracle_export_dm.dmp. You can change this to a more meaningful name by clicking **Update** and entering the name you prefer. If you remove the .dmp file extension, the system adds it.

Oracle Export Data Mart Parameters

The Oracle Export Utility requires the following parameters. More information is available in the Oracle Export Utility documentation.

> **Note:**
>
> Because these are predefined, required Parameters you should change only the default value, Read Only setting, or Prompt for each Parameter or the Data Mart may not function properly; see Setting Data Mart Parameter Values.

- **Compress:** Should Oracle Export export data into a single extent? The default value is Y.
- **Statistics:** Specifies the type of database optimizer statistics to generate when the exported data is imported. Options are ESTIMATE, COMPUTE, and NONE. The default value is ESTIMATE.

Oracle Export Data Mart Planned Outputs

For Oracle Export Data Marts, the system creates one Planned Output for the whole Data Mart to contain data from all Table instances mapped to the Data Mart instance's Table Descriptors. The Planned Output name is always Oracle Export DM Export File. The file name is *filename_attribute_value*.dmp.

The system also creates a Planned Output for the log file. All Oracle Export log files have the same name: oracle_export.log.

You can classify a Planned Output in order to classify the actual output when it is generated; see Classifying Outputs for further information.

Installing Data Mart Instances

You can install a Data Mart instance directly from its Properties screen, using the Install button, or in its Work Area (see Installing a Work Area and Its Objects).

When you install a Data Mart instance using the **Install** button on its Properties screen:

- The system checks in the Data Mart instance and definition and also the Table instances in the current Work Area to which the instance is mapped.
- The system attempts to install the Data Mart instance and the Table instances to which it is mapped. The system displays a success or error message. If the installation fails, the error message displays the name of any objects that were not installable.

> **Note:**
>
> If the Data Mart definition or any of the Table instances to which the Data Mart is mapped is not installable, the system cannot install the Data Mart instance. See Installation Requirements for Each Object Type for the reasons these objects may not be installable.

> **Note:**
>
> To continue working on the Data Mart, check it out.

- Access the Log File

Access the Log File

To see the log file for the installation, you must go to the Work Area Installation screen, as follows:

1. Click the **Applications** tab. The main Application Development screen opens.
2. Click the name of the Work Area you are working in. The Work Area screen opens.
3. From the **Actions** drop-down list, select **Installation History**.
4. Click **Go**. The system displays the Installation History screen with the log files in chronological order.
5. Click the **View Log** link for the most recent installation attempt or for the date and time that you ran the install process. The system displays the log file.

For information on installation and on reading the log file, see Installing a Work Area and Its Objects.

Modifying Data Marts

If you have the necessary privileges, you can modify a Data Mart definition either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area. In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition as a whole, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Data Mart Instance Properties
- Modifying Data Mart Definition Properties

Modifying Data Mart Instance Properties

On the Data Mart instance's Properties screen, click **Update** to enter changes. Oracle LSH creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must install the new version for the changes to take effect.

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field applies to the instance only. It specifies the Data Mart definition to which this Data Mart instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Table Descriptors, Parameters, and Planned Outputs.
  - Any new Table Descriptors are not mapped.
  - The Data Mart's status changes to **Non Installable**.

If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.

> **Note:**
>
> It is possible to select a new source definition of a different Data Mart type. Exercise caution when selecting a different definition.

Modifying Data Mart Definition Properties

You can go to a Data Mart definition's Properties screen in one of the following ways:

- **From the Data Mart's Properties screen**: Click the hyperlink of the Data Mart definition that appears in the Definition field.
- **From the Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Data Mart definition screen, click **Update** to enter changes. Oracle LSH creates a new version of the definition. You can change the following properties:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.

  You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.
- Modifying Table Descriptors
- Modifying Attributes and Parameters
- Modifying Planned Outputs

Modifying Table Descriptors

In the Data Mart definition you can remove existing Table Descriptors and add different ones.You must check out the definition to add, remove, or update Table Descriptors, but not to map, unmap, or remap Table Descriptors.

In the Data Mart instance, you can change the Table Descriptor mappings, which are part of the Data Mart instance, not the definition; see Mapping Table Descriptors to Table Instances . You do not need to check out the definition to modify the mappings.

Modifying Attributes and Parameters

You cannot add or remove Attributes or Parameters because they are predefined for each Data Mart type. Attributes and Parameters belong to the Data Mart definition. You must check out the definition to update Parameters. In the Data Mart definition you can change Attribute values and, for Parameters, the default value and Read Only and Prompt settings. You should not change anything else or the Data Mart may not function properly.

To change Parameter values at the instance level you can use the Execution Setup. To make your settings available to other instances of the same Data Mart, allow the

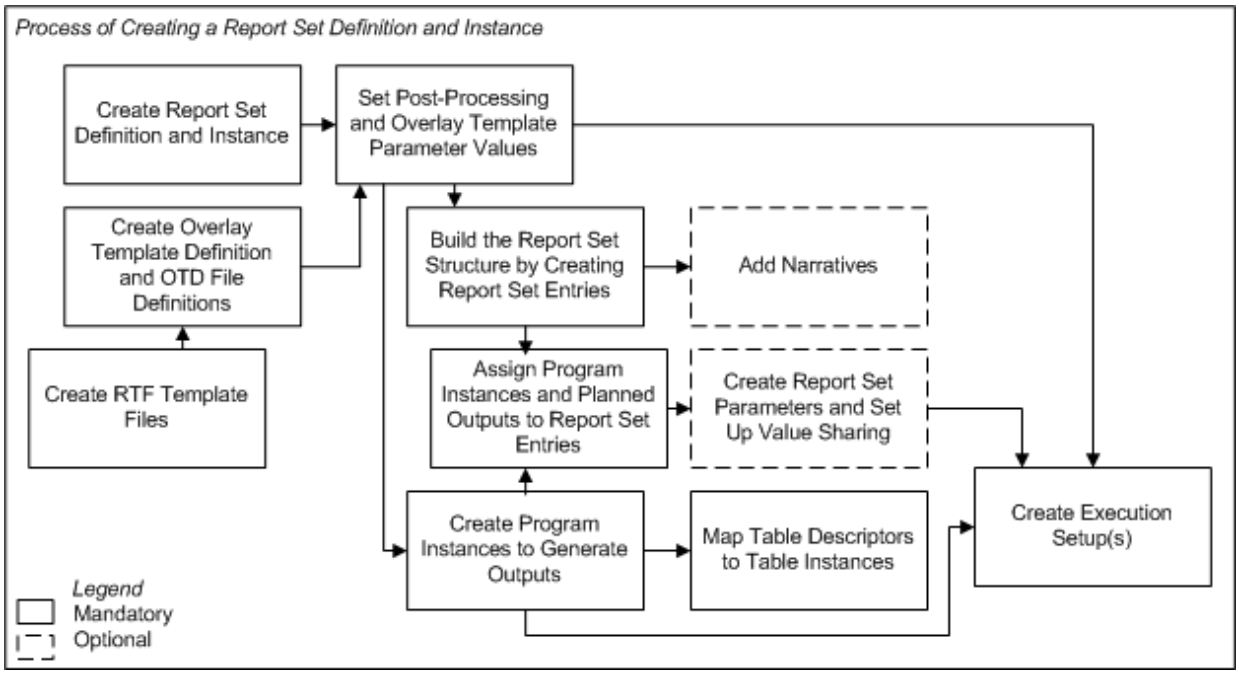
Execution Setup to be used as a Template. Select **Execution Setups** from the **Actions** drop-down list in the Data Mart instance in the Work Area. See Creating, Modifying, and Submitting Execution Setups.

Modifying Planned Outputs

Planned Outputs are predefined for each Data Mart type and belong to the Data Mart definition. You cannot modify Data Mart Planned Outputs except to change their classifications, which affect the classifications of the actual Data Mart output. See Classifying Outputs for further information.

9
Defining Report Sets

Figure 9-1 Process of Creating a Report Set Definition and Instance



This section contains information on the following topics:

- About Report Sets
- How to Work on a Report Set
- Creating Overlay Templates
- Creating a Report Set
- Using the Report Set Properties Screen
- Using the Report Set Structure View
- Creating and Setting Report Set Parameters
- Defining Report Set Entries
- Defining Programs to Generate Reports
- Installing Report Sets
- Validating Report Set Definitions and Outputs
- About Report Set Planned Outputs
- Modifying Report Sets

About Report Sets

A Report Set provides a way to organize, manage, and run a set of reports together under version control. You can define numbered chapters, sections within the chapters, and any number of levels of subsections, with reports and/or narrative text at any level. When you choose to use postprocessing, the Oracle Life Sciences Data Hub (Oracle LSH) generates a table of contents and hyperlinks to each report.

Each report in a Report Set is generated by the execution of a Program instance whose corresponding Planned Output is assigned to the particular chapter, section, or subsection (all of which are called Report Set Entries) where the report appears in the final output. A single Program instance can generate multiple reports, each of which can be assigned to a different Report Set Entry. Reports can take the form of tables, listings, figures, or text, depending on the technology you use to create them and how you define them.

You can include Program instances of different technology types—including SAS, Oracle Reports, and PL/SQL—in the same Report Set. When you execute the Report Set, the system sends each Program instance to the appropriate engine for execution and integrates the results.

Postprocessing

Oracle LSH includes Oracle XML Publisher for use in an optional postprocessing step for Report Sets. During postprocessing, after the system has generated all the reports, Oracle XML Publisher generates a single PDF file for the entire Report Set (or breaks the Report Set into volumes if you choose), paginates and concatenates the individual reports, numbers the pages, and creates PDF bookmarks.

To use the postprocessing feature you must create reusable Overlay Template RTF files that include publishing specifications such as multilayered page layouts, graphics, and watermarks for each type of page in your Report Set (for example, the table of contents, listings, and figures, in landscape and portrait orientations). You upload these RTF template files to an Overlay Template File, which is part of an Overlay Template definition in Oracle LSH.

You can execute a Report Set without the postprocessing step to save time while you are developing and testing the Report Set and its Program instances. In that case the system generates an output for the Planned Output assigned to each included Report Set Entry, with the file type specified for the Planned Output. The Post-Processing Parameter called Post Process controls whether or not postprocessing is included in a particular execution.

If you plan to use postprocessing you must define the file type of all Planned Outputs assigned to Report Set Entries as PDF.

Parameter Value Propagation

You can set up value sharing among Parameters to promote consistency and reduce the work required to submit the Report Set for execution.

For example, many Program instances may have a Parameter for Study Name or Study ID. You can define a Report Set Parameter to collect the Study Name or ID at runtime at the top level of the Report Set, and set up value sharing so that the person submitting the Report Set for execution enters the value for Study Name or ID once,

and the system automatically propagates that value to the Study Name or ID Parameter of all the Program instances you specify.

You can also use an output Parameter value of one Program to populate the value of an input Parameter of another Program. See Setting Up Parameter Value Propagation for further information.

Concurrent Editing

Multiple people can simultaneously modify a single Report Set. The first person to modify it must check out the definition, but while it is checked out, anyone else with Modify privileges on the Report Set definition can modify it.

However, only one person can work on any one section at a time; see Concurrent Editing of Report Sets for further information.

> **Note:**
>
> Shared editing works only in a Report Set instance that is contained directly in a Work Area. It does not work in a Report Set instance that is contained in a Workflow.

Execution

Oracle LSH does not necessarily execute a Report Set in the order displayed in the Structure view, but takes dependencies into account. If you have set up value propagation from the output Parameter of one Program to the input Parameter of another Program, the system executes the Program that produces the output value before the Program that takes the input value.

The log file the system generates each time it runs a Report Set contains detailed information about each stage of the Report Set execution.

Report Set submissions include two jobs, both of which are displayed on your My Home page. The first job starts the second, which is a batch process.

Partial Execution

Using the Execution Setup for the Report Set as a whole you can submit the whole Report Set or specify one or more Report Set Entries to execute. You can also select a single Report Set Entry in the Structure view and submit it with or without Report Set Entries under it in the Report Set hierarchy. You can also submit a single Program and the system will produce the Report Set Entries to which it is assigned.

Validation of Individual Report Set Entries

You can validate different sections of the Report Set at different times. See Validating Report Set Definitions and Outputs for details.

Reports on Report Set Definitions and Instances

From the **Actions** drop-down list, you can generate reports that provide information on a Report Set definition or instance; see System Reports for information.

How to Work on a Report Set

Report Sets can be very large and complex, and many people can work on them at the same time; see Concurrent Editing. To make Report Set development work best, use the following general process:

1. **Create overlay template files and Overlay Template Definitions**. To produce a unified PDF output of your Report Set you must create RTF templates, upload them to Oracle LSH, and define Overlay Template Definitions (OTDs) in Oracle LSH. You must define OTDs before you can set values for the predefined Overlay Template Parameters to specify which template(s) to use for each Report Set Entry, or branch, of the Report Set.

   You can use the same OTDs and OTD files for many Report Sets. See Creating Overlay Templates and Setting Overlay Template Parameter Values for further information.

2. **Create the Report Set**. Create the Report Set definition and an instance of it, then click **Update** to set the Strict and Unique numbering properties for the Report Set. See Creating a Report Set for further information.

3. **Set Parameter values at the top Report Set level**. Set values at the top Report Set level for predefined Overlay Template and Post-Processing Parameters. This ensures that all Report Set Entries have values for these Parameters in the Report Set instance and in the Execution Setup. Creating and Setting Report Set Parameters.

4. **Create Report Set Parameters (optional)**. To make the Report Set easier to submit and to help ensure data consistency, you may want to define one or more Parameters at the Report Set level. For example, if all the reports in a Report Set are about the same study, and the Programs that create the reports have an input Parameter to identify the study, you may want to define a Study Parameter at the Report Set level and share its value with each Program's Study Parameter. The Report Set Study Parameter appears at the top of the Report Set's Execution Setup where the user can easily find it and set it once for the whole Report Set. See Setting Up Parameter Value Propagation for further information.

5. **Define the Report Set structure**. A Report Set has top-level chapters that contain subchapters that can in turn contain any number of levels of sections and subsections. All of these are called Report Set Entries. You must define a Report Set's structure by defining hierarchical Report Set Entries from the top down; see Using the Report Set Structure View.

6. **Develop, test, and validate Report Set Entries and Programs**. Develop the Programs that generate the Report Set's reports. Assign each Program Planned Output to the Report Set Entry (chapter, section, or subsection) where it should appear. Add pre- and post-report narratives as necessary to Report Set Entries. Adjust the settings for Post-Processing and Overlay Template Parameters as necessary. Test and validate Programs, Report Set Entries, and report outputs according to your company's standards. See Defining Report Set Entries and Validating Report Set Definitions and Outputs for further information.

Concurrent Editing of Report Sets

More than one person can work on a Report Set at the same time. After one person checks out the Report Set, any other user with Modify privileges on the Report Set can

also work on it. In order to protect each person's work, the system enforces the following behavior:

> **Note:**
>
> If you try to make a change that is not allowed because of the activity of another person, you get an error message with the username of the person whose work is conflicting with yours. In most cases you can click Apply again immediately and the system applies your change.

- **Report Set Structure**. Many people can create, move, copy, modify, or remove Report Set Entries as long as they are all working in different branches of the Report Set.

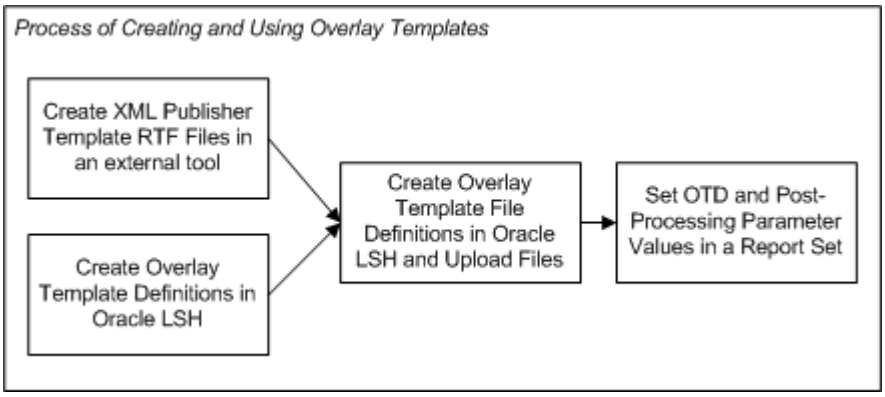
  One person cannot move, reorder, copy, or remove a Report Set Entry that is a parent in the same hierarchy (branch) of a Report Set Entry that another person is currently modifying.
- **Report Set Entry**. Only one person at a time can modify a particular Report Set Entry's properties. Different people can modify different Report Set Entries in different branches of the Report Set hierarchy.
- **Program Instance**. Only one person at a time can modify a particular Program instance contained in the Report Set. Different people can modify different Program instances.
- **Install Program Instance**. One person can install a single Program instance in the Report Set at any time without impacting the work of other people on the Report Set. Installing a Program instance implicitly checks in the Report Set definition and instance, installs the Report Set instance, and checks out the Report Set definition and instance again.
- **Install Report Set**. Installing a Report Set from a Work Area is permitted but not recommended during concurrent editing. Oracle recommends installing a Program instance from the Report Set Structure view as a means of installing the Report Set structure during concurrent editing.

  During installation of a Program in a Report Set, other people can continue to work on the Report Set definition through the instance, including making changes to the Report Set structure and Programs. However, noone can work on the Report Set instance during installation of the Report Set in a Work Area. This includes mapping Program instances. See Installing Report Sets for further information.

  > **Note:**
  >
  > If the person running the Work Area installation has Checkin Administrator privileges, all Program instances are checked in without warning, including those checked out by other users.

- **Execute Report Set**. One person can execute a Report Set or a portion of a Report Set without impacting the work of other people.

Creating Overlay Templates



This section contains the following topics:

- About Overlay Templates
- Creating Template Files
- Creating an Overlay Template Definition
- Creating an Overlay Template File Definition

About Overlay Templates

Oracle LSH's postprocessing feature uses Oracle XML Publisher to generate a single- or multi-volume PDF-format Report Set with a unified Table of Contents from the individual PDF reports and narratives generated by the first stage of Report Set execution.

To use the postprocessing feature, you must design and create RTF templates that can include graphics, watermarks, hyperlinks, text variables, and the font styles and sizes you prefer.

You create template files in Microsoft Word or a similar tool, save them as RTF files, and upload each one to an Overlay Template Definition (OTD) File definition in an OTD. In the OTD File definition you specify the file's orientation, paper size, language, and rotation.

> **Note:**
>
> For information on creating template files, see the *Oracle XML Publisher User's Guide*, which is available on My Oracle Support and the Oracle Technology Network.

Templates Applied at Runtime

For the Table of Contents and Narratives, Oracle LSH uses the runtime values of the Post-Processing Parameters Paper Size, Language, and Orientation to determine

which template files associated with the OTD to apply. To determine which template files to use for the actual reports, or content, Oracle LSH uses Paper Size, Language, and the orientation of the actual PDF report output.

Therefore you can, for example, create a single OTD with template files for both A4 and US letter-sized paper, and use different values for the Paper Size Post-Processing Parameter to print it out on A4 paper in Europe or Japan and letter-sized paper in the US or Canada.

> **Note:**
>
> In Oracle LSH Release 2.5 the only language supported is US English.

Overlay

You can create multiple template files in the same Overlay Template definition with the same values for Paper Size, Orientation, and Language. In this case, Oracle XML Publisher uses all of them, overlaying them on top of each other on each page. For example, you can create one template file with a rotation of 90 degrees to print the Report Set title and your company logo vertically along the side margin, and another template file for the main body of the page with a rotation of 0. Because both files have the same orientation and paper size, the system applies both; see Figure 9-2.

Creating Template Files

Oracle recommends creating a single OTD containing all the template files you may need to postprocess a single Report Set. Decide which options represented by Post-Processing Parameters you want to support—A4 and/or US Letter-sized paper, landscape and/or portrait orientation— and create the template files necessary to support them in a single OTD. The values for the Post-Processing Parameters can be set in the Report Set definition, in the Execution Setup definition, or by a user submitting the Report Set for execution. Oracle LSH selects the template(s) to use for each section of the Report Set based on these Parameter values at runtime.

OTD File Types

You must create at least one RTF template file for each OTD File type that you plan to use: Coversheet, Table of Contents, Content, Narrative, Narrative Content, Page Numbering, TOC Overlay, and In Progress; see Creating an Overlay Template File Definition for a description of each file type.

Text Variables

You can use text variables in your template files to substitute an actual value from the Report Set at runtime. These variables are sensitive to context within the Report Set. For example, if you use the text variable `<?xdo66:title?>` it displays the title of the Report Set when used in the Coversheet or Table of Contents template, and it displays the Report Set Entry title when displayed in a Narrative or Content template for a chapter or section.

You can also use text variables to display the runtime value of user-defined Parameters; for example, `<?study?>`. The variable displays the value of the Study Parameter appropriate for the context: Report Set, Report Set Entry, or Program.

Footers

You can include a footer in any of the OTDs if required. Enter the following variable in the OTD RTF's footer: `<?/xdobb:item/footer?>`.

Creating an Overlay Template Definition

You create Overlay Template definitions directly in an Application Area or Domain library, not a Work Area. They do not have instances.

To create an Overlay Template, do the following:

1. In a Domain or Application Area, click **Manage Definitions**.
2. From the **Create** drop-down list, select **Overlay Template**. The Create Overlay Template screen opens.
3. Enter a **Name** for the Overlay Template.
4. Enter a **Description** for the Overlay Template.
5. From the **Default Paper Size** drop-down list, select one of the following:
   - **A4**. The standard paper size used in Europe and Japan: 21 x 29.5 cm.
   - **US Letter**. The standard paper size used in North America: 8.5 x 11 inches.

   This value serves as the default value for the corresponding Parameter of each Overlay Template File. You can change the value for any Overlay Template File definition as necessary.

   > **Note:**
   >
   > If you have some template files for A4 paper and others for US Letter-sized paper, you can set this value to A4 first, for example, upload all the template files for A4 paper, then change the default to US Letter and upload all the template files for US Letter-sized paper.

6. From the **Default Language** drop-down list, select a language. In Oracle LSH Release 2.5 the only option is US English.

   This value serves as the default value for the corresponding Parameter of each Overlay Template File in the Overlay Template.
7. Classify the Overlay Template definition; see Classifying Objects and Outputs for further information.
8. Click **Apply**. The Overlay Template Definition Properties screen appears.
9. Click **Add** to add Overlay Template File definitions (see Creating an Overlay Template File Definition below).

Creating an Overlay Template File Definition

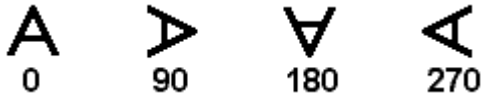
An Overlay Template Definition (OTD) File is an Oracle LSH definitional object contained in an OTD. It consists of an uploaded XML template—an RTF file that contains formatting design information (see Creating Template Files) and a set of Parameters that serve as labels.

To create an Overlay Template Definition File, do the following:

1. In the Overlay Template Definition Properties screen, click **Add** in the Overlay Template Definition Files section. The Overlay Template Definition File screen opens.
2. Click the **Browse** button to find the RTF template file you want to upload. The system opens a standard Browse window.
3. Navigate to the location of the file you want to upload, select the file, and click **Open**.
4. Select a **Type**:
   - **Content**. Files of type Content determine the appearance of the actual Report Set reports. This is the default Type.
   - **Coversheet**. Files of type Coversheet determine the appearance of the Report Set's title page.
   - **In Progress**. Create one OTD file of type In Progress. When the system cannot execute a Report Set Entry because its associated Program instance is checked out or fails execution, the system uses the In Progress OTD file as a placeholder for the unexecuted report.
   - **Narrative**. Files of type Narrative are used to display the pre- or post-narratives for the report.
   - **Narrative Content**. Create and upload a PDF file (usually blank) for Oracle XML Publisher to use to format the text content for the narrative specified in the Report Set definition.
   - **Page Numbering**. Oracle XML Publisher requires a special OTD file for page numbering because it must generate a page number dynamically for each page. There is an XML tag that for the position and format of the page number. See the Oracle XML Publisher documentation for further information.

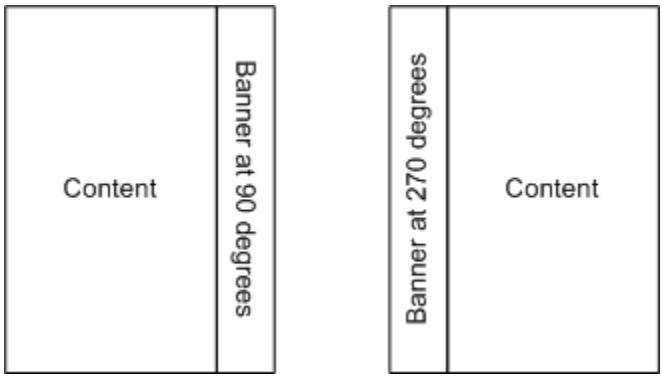
     To produce a Report Set with page numbers, you must define and upload one OTD file of the Page Numbering type for each paper size/orientation combination represented in this Overlay Template Definition.
   - **Table of Contents**. Files of type Table of Contents determine the appearance of the Report Set's table of contents.
   - **TOC Overlay**. If necessary, use this type to provide a header and footer or other overlay formatting for the table of contents.
5. Select a printing **Orientation**:
   - **Portrait**. Portrait files must be printed vertically.
   - **Landscape**. Landscape files must be printed horizontally.

   The default value is Portrait.
6. Select a **Paper Size**. The default value comes from the Paper Size Parameter of the Overlay Template definition.
   - **A4**. The standard paper size used in Europe and Japan: 21 x 29.5 cm.
   - **US Letter**. The standard paper size used in North America: 8.5 x 11 inches.
7. Select a **Language**. In the Oracle LSH Release 2.5 there is only one option: **US English**.
8. Select a **Rotation** (in number of degrees):



For example, you can use a rotation of 90 or 270 degrees to create a banner along one side of a page. The figure below shows the direction in which text appears when you define two template files with a partite orientation and the same paper size. The Content template has a rotation of zero (0) and the banner template file has a rotation of 90.

Figure 9-2 Example with Banner Template File



> **Note:**
>
> Do not change the setting for Orientation (for example, from landscape to portrait) simply because you are using a 90- or 270-degree rotation. The rotated file should have the same Orientation setting as the other files with which it is to be used. Further information is available in the *Oracle XML Publisher User's Guide*.

Creating a Report Set

When you create a Report Set in a Work Area, you are actually creating an instance of a Report Set definition.

To create a new Report Set instance:

1. In a Work Area, select **Report Set** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Report Set screen.
3. Choose one of the following options:
   - **Create a new Report Set definition and instance**. Choose this option if no Report Set definition exists that can meet your needs, either as it is or with some modification.

- **Create an instance from an existing Report Set definition**. Choose this option if a Report Set definition already exists that meets your needs.

  If you can adapt an existing Report Set definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.

4. Depending on your choice, follow one of the following sets of instructions:

- Creating a New Report Set Definition and Instance
- Creating an Instance of an Existing Report Set Definition

Creating a New Report Set Definition and Instance

When you select **Create a new Report Set definition and instance** in the Create Report Set screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects. The system uses the Name internally. It must be unique within the container.
   - **Title**. The system enters the value you entered for the Name as the default value. You can change it. It is the Title, not the Name, that appears on the Report Set output. The Title is not required to be unique.
   - **Description**. See Creating and Using Object Descriptions.
2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Report Set.

   The system opens the Properties screen for the new Report Set instance with the Report Set definition checked out. The Report Set properties **Strict Numbering** and **Unique Numbering** are both set to **Yes** by default. If you want to change their values, you must go to the Report Set definition and change them there; click on the Definition hyperlink in the Instance Properties section.

   The properties are:

   - **Strict Numbering**. If set to **Yes**, Report Set Entries must be numbered sequentially, with no gaps. The system automatically generates sequential numbers. This is the default setting.

     If set to **No**, gaps are allowed in the numbering of Report Set Entries. The system generates sequential numbers by default, but you can override the entry number and if you remove a Report Set Entry the system does not renumber the remaining Report Set Entries to close the gap (however, you can do this manually).
   - **Unique Numbering**. If set to **Yes**, each Report Set Entry must have a number that is different from the number of all other Report Set Entries in the same Report Set. The system automatically generates unique numbers.

If set to **No**, unique numbers are not required. The system generates unique numbers by default, but you can override the entry number to create duplicate numbers.

Table 9-1 Possible Combinations or Strict and Unique Numbering Settings

Unique Numbering	Strict Numbering	Automatic Renumbering Possible?	Valid Example	Invalid Example
Y	Y	Yes	14.2.1 Table 14.2.2 Listing 14.2.3 Table	14.2.1 Table 14.2.1 Listing 14.2.3 Table
Y	N	No	14.2.1 Table 14.2.3 Listing 14.2.6 Table	14.2.1 Table 14.2.1 Listing 14.2.3 Table
N	Y	No	14.2.1 Table 14.2.1 Listing 14.2.2 Table	14.2.1 Table 14.2.1 Listing 14.2.3 Table
N	N	No	No restriction	No restriction

Creating an Instance of an Existing Report Set Definition

Use this option if there is an existing Report Set that you can use exactly as it is. If you use an existing Report Set as a definition source, you can use all the Report Set Entries, Program instances, Planned Outputs, Execution Templates, Post-Processing and Overlay Template Parameter values, and Parameter value propagation that are already defined. See Creating an Instance of an Existing Definition for instructions.

After you have created the Report Set instance, you must map the Table Descriptors of all the Program instances contained in it to Table instances; see Mapping Table Descriptors to Table Instances for instructions.

Using the Report Set Properties Screen

If you are working in a Work Area, you see the properties of both the Report Set instance and the Report Set definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See Modifying Report Sets for information on modifying Report Sets.

See also Figure 9-1.

This section contains the following topics:

- Instance Properties
- Definition Properties
- Buttons

Instance Properties

You can see the following instance properties:

Name: You can click **Update** and modify the name. See Naming Objects for further information.

Description: You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.

Definition: This field specifies the Report Set definition to which this Report Set instance points.

To upgrade to a new version of the same definition, use the **Upgrade to Latest** button. See Upgrading to a Different Definition Version from an Instance.

Version: This field displays the current version number of the Report Set instance.

Version Label: This field displays the version label, if any, for the current Report Set instance version.

For further information on object versions, see Understanding Object Versions and Checkin/Checkout.

Validation Status: This field displays the current validation status of the Report Set instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.

Status: This field displays the install status of the Report Set: installable or noninstallable. See Installation Requirements for Each Object Type.

Definition Properties

Checked Out Status: This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to modify the Report Set structure or to add, remove, assign, or reassign Program instances. You can change Table Descriptor mappings without checking out the definition. See About Programs in Report Sets and Understanding Object Versions and Checkin/Checkout for further information.

Latest Version: If set to **Yes**, this Report Set instance is pointing to the latest version of the Report Set definition. If set to **No**, this Report Set instance is pointing to an older version of the Report Set definition.

View Latest: You can see this button only if the current Report Set instance does not point to the latest definition version. Click this button to view the latest Report Set definition.

Upgrade to Latest: This button is grayed out if the current Report Set instance already points to the latest Report Set definition. Click this button to upgrade the current Report Set instance to the latest definition version. For more information on upgrading instances, see Upgrading Object Instances to a New Definition Version.

Checked Out By: This field displays the username of the person who has the Report Set definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.

Version Label: This field displays the version label, if any, for this definition version.

Strict Numbering: If set to **Yes**, numbers must be sequential, with no gaps. If set to **No**, gaps are allowed. You must be in the Report Set definition in its Application Area or Domain to change this value.

Unique Numbering: If set to **Yes**, numbers must be unique within the parent. If set to **No**, duplicate numbers are allowed. You must be in the Report Set definition in its Application Area or Domain to change this value.

Title: This field displays the Report Set definition title.

Validation Status: This field displays the current validation status of the Report Set definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Report Set in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.

Status: This field displays the installable status of the Report Set: Installable or Non Installable. See Installation Requirements for Each Object Type.

Buttons

From a Report Set instance in a Work Area, you can use the following buttons:

Submit

Click **Submit** to run the Report Set instance. Before you can run the Report Set, you must install it and create an Execution Setup for it (select **Execution Setups** from the **Actions** drop-down list).

Update

Click **Update** to modify the Report Set instance properties. See Modifying Report Set Instance Properties.

Check In/Out and Uncheck

Click these buttons to check out, check in, or uncheck the Report Set definition. Different buttons are displayed in the Report Set Definition Properties section depending on the checked out status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of the person who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.

View Latest/Upgrade to Latest

If the definition is not the latest version, you can click to view the latest version and upgrade to the latest version if you want to. See Upgrading to a Different Definition Version from an Instance.

Using the Report Set Structure View

This section contains the following topics:

- Navigating to the Report Set Structure View
- Building and Modifying the Report Set

Navigating to the Report Set Structure View

To view, build, or modify a Report Set's structure, do the following:

1. Navigate to the Report Set instance in its Work Area in the **Applications** tab.
2. Click the Report Set name's hyperlink. The Report Set opens.
3. If it is not already showing, select **Structure** from the **View** drop-down list.

Building and Modifying the Report Set

The Report Set's Structure view, with **Structure** selected in the **View** drop-down, shows the structure of the Report Set in a display similar to a table of contents.

Each chapter, section, and subsection is called a Report Set Entry. You can define any number of levels of Report Set Entries. You must define Report Set Entries from the top level down, creating each new Report Set Entry (RSE) as a child of an existing Report Set Entry or of the Report Set itself.

This section contains the following topics:

- Modifying the Report Set
- Report Set Entry Information Displayed

Modifying the Report Set

In the Report Set Structure view you can build and modify the Report Set structure and do many other things as follows:

1. Select the Report Set itself or a Report Set Entry (RSE).
2. From the **Select and** drop-down list, select the action you want to perform on the selected Report Set or Report Set Entry.

   Use **Add RSE**, **Remove RSE**, **Reorder**, **Copy**, and **Move** to create and modify the Report Set structure. Details for all options are given below.
3. Click **Go**.

The following options are available:

- Add Program
- Add RSE
- Assign Planned Output
- Copy
- Default Execution Setup
- Execution Setup
- Install Program
- Map
- Move

- Quick Submit
- Remove RSE
- Reorder
- Submit
- View All Outputs
- View Output

Add Program

The Create Program screen opens and you can either create (add) an instance of an existing Program definition or create a new Program definition and instance. If you add an instance of an existing Program that has at least one primary Planned Output, the system assigns the first primary Planned Output to the selected Report Set Entry. You can change the assignment if necessary by using the Assign Planned Output option. See Defining Programs to Generate Reports for information about Programs inside Report Sets and Creating a Program for general instructions.

Add RSE

The Add Entries screen opens and you can add one or more Report Set Entries under (at the child level of) the one you selected or modify existing Entries; see Creating Multiple Report Set Entries for instructions. If the selected Report Set or Report Set Entry already contains one or more child Report Set Entries, the system adds the new ones below the existing ones.

After you have added Report Set Entries you can define additional details; see Setting Report Set Entry Properties and assign a Planned Output to the Report Set Entry; see Assign Planned Output below.

Assign Planned Output

The Assign Program Instance and Planned Output screen opens. Do the following:

1. Click the Search icon to search for a Program already assigned to the Report Set.

> **Note:**
>
> Only checked out Program instances can be assigned to a Report Set Entry.

2. Select the Planned Output you want to assign.
3. Click **Select**.

If you select a Report Set Entry that already has a Planned Output assigned, you can change the assignment by selecting a different one.

Copy

You can copy a Report Set Entry and its child Report Set Entries into another location in the same Report Set or a different one. The Copy operation includes all child Report Set Entries and the Execution Setup fragment for the selected Report Set Entry and all

its children (if such Execution Setup fragments exist). You have the option to copy Program instances assigned to Report Set Entries being copied.

1. Select the Report Set Entry you want to copy and then select **Copy** from the drop-down list.
2. Click **Go**. A message appears asking if you want to include Program instances assigned to the Report Set Entry (and its children).
   - Click **No** if you do not want to copy Program instance(s).
   - Click **Yes** to copy Program instance(s).
3. Select the Report Set into which you want to copy the Report Set Entry; see Pasting Objects.

If you choose to copy Program instances, the system copies their Execution Setups and Table Descriptor mappings and maintains the existing links to Program definitions and their Planned Outputs.

If there is already a Program with the same name in the target Report Set, the system creates the copy with the name `Copy Of` *`Program_Name`*. When you copy a Report Set Entry into the same Report Set, this always happens. If you prefer to have a single instance of the Program in the Report Set, so that the Program is executed only once during Report Set execution, do one of the following:

- Remove the copied Program from the Report Set and assign the original Program to the Report Set Entry.
- Unassign the Program before you do the Copy operation and reassign it afterward.

> **Note:**
>
> You cannot select the Report Set's top level and then select **Copy** from the drop-down list. You can copy a whole Report Set from the Applications screen but not from the Report Set screen.

> **Note:**
>
> To copy multiple Report Set Entries at the same level, click the hyperlink of the parent Report Set Entry, then select and copy the Report Set Entries.

Default Execution Setup

View and modify the default Execution Setup for selected Report Set instance or Report Set Entry and its child Report Set Entries; see Report Set and Workflow Execution Setups.

If no Execution Setups are defined, the system generates a default Execution Setup and opens it.

Execution Setup

The system displays all Execution Setups defined for the selected Report Set instance. The list is the same regardless of which Report Set Entry you select; you see the Execution Setups for the Report Set instance as a whole. You can then click on the hyperlink of any Execution Setup and navigate to any Report Set Entry to modify its settings.

If no Execution Setups are defined, the Create Execution Setup screen opens and you can create one.

Install Program

Install the Program instance assigned to the selected Report Set Entry. If the Program instance is not installable you get an error message.

When you install a Program instance from the Report Set Structure view, the system also checks in the Report Set definition and instance and installs the Report Set instance, then checks the Report Set definition and instance back out. You can use this functionality to install changes to the Report Set structure even if no Program instances need to be installed.

Map

You can map or remap (using Automatic Mapping by Name) the Table Descriptors of all Programs assigned to the selected Report Set Entry and all its child Report Set Entries at the same time. If you select the top level of the Report Set you can map the Table Descriptors of all Programs in the Report Set. For further information, see Automatic Mapping by Name. For other mapping options, see About Programs in Report Sets.

> **Note:**
>
> The Report Set instance is not installable until all the Table Descriptors of all the Program instances it contains are mapped.

Move

You can move a Report Set Entry into another location in the same Report Set or a different one. The Move operation includes all child Report Set Entries and the Execution Setup fragment for the selected Report Set Entry and all its children.

If you move a Report Set Entry within the same Report Set, the system maintains the existing links to Program instances and their Planned Outputs.

If you move the Report Set Entry to a different Report Set, a message appears asking if you want to include Program instances assigned to the Report Set Entry and its children. If you choose to include Program instances, the system copies all Program instances assigned to the Report Set Entries you are moving, and their Execution Setup fragments (if any) and recreates the Program instance and Planned Output assignments in the new location.

See Pasting Objects.

> **Note:**
>
> Program instances are never removed implicitly from a Report Set, even if all the Report Set Entries to which they are assigned are moved or removed.

> **Note:**
>
> You cannot select the Report Set's top level and then select **Move** from the drop-down list. You can move a whole Report Set from the Applications screen but not from the Report Set screen.

Quick Submit

The system submits the selected item—either a Report Set Entry and all its children, the entire Report Set, or a Program instance—using the default Execution Setup with its default values, without opening the Execution Setup screen. The system upgrades the Execution Setup if necessary and validates it. The submission fails if:

- The Execution Setup cannot be upgraded; for example, if another user is currently modifying the default Execution Setup
- The Execution Setup is invalid; for example, the Parameters that apply to the portion of the Report Set being submitted have invalid values

In the Job Execution section of the My Home page you see two jobs. A temporary job starts the actual report generation job.

Remove RSE

The system deletes the Report Set Entry you select. If you select a Report Set Entry that contains other Report Set Entries, the system removes them also.

> **Note:**
>
> If you try to remove a Report Set Entry to which other Report Set Entries have links, you receive a warning. If you choose to continue, the system removes the selected Report Set Entry and all links between it and other Report Set Entries. The system does not remove any Program instances assigned to any Report Set Entries being removed.

> **Note:**
>
> You cannot select the Report Set's top level and then select **Remove** from the drop-down list. You can remove a Report Set from the Applications screen but not from the Report Set screen.

Reorder

Select the Report Set or Report Set Entry whose child Report Set Entries you want to reorder; see Reordering and Renumbering Objects.

Submit

The system displays the Submit screen based on the default Execution Setup for the selected Report Set instance. If you select a Report Set Entry and then Submit, the Report Set Entry you selected, and all its children, are selected for inclusion in the execution. You must define an Execution Setup and install the Report Set instance before you can submit any part of it for execution.

View All Outputs

The system displays a list of all outputs produced for the selected Report Set or Report Set Entry. The system displays the following information for each output: Output Validation Status, Creation TS (Timestamp), Creation User (the username of the person who ran the job that created the output), Job ID, Program Instance Name, Program Instance Version, Path to Executable (Program) Instance, and Title.

In addition, there is an icon in the View column that you can click to view the output itself.

View Output

The system displays output properties information about the current output produced for the selected Report Set or Report Set Entry, including: title, file name, job ID, execution status, description, validation status, execution user (the username of the person who executed the job), job start time, blinding status, and primary output (yes/no).

To view the output itself, click **View File**.

> **Note:**
>
> Oracle LSH does not allow removing, copying, moving, or reordering a Report Set Entry if it is assigned to a Program instance that is checked out.

Report Set Entry Information Displayed

The Report Set Structure view displays the following information about each Report Set Entry and the Report Set as a whole:

- **Full Title** is the Report Set Entry's concatenated title that includes its Entry Number Prefix, Parent Number, Delimiter, Entry Number, Entry Number Suffix, and Title, in that order.
- **VS (Validation Status)** refers to the standard validation status of the Report Set Entry; either Development (Dev), Quality Control (QC), or Production (Prod).
- **Summary Output VS (Validation Status)** is the calculated validation status derived from outputs' and child Report Set Entries' validation statuses. The

possible values are: Null, Not Assigned, N/A (Not Applicable), Development, Quality Control, or Production; see Summary Output Validation Status.

- **Output Creation TS (Timestamp)**. The system displays the timestamp of the creation of the current output.
- **Narrative**. If the checkbox is checked, there is a Pre- and/or Post-Narrative assigned to the Report Set Entry.
- **Assigned Program Instance**. If a Program instance is assigned to the Report Set Entry, the system displays a link to its Properties screen.
- **Program Instance VS (Validation Status)**. If a Program instance is assigned to the Report Set Entry, the system displays its validation status. You can click the hyperlink to change its validation status if you have the required privileges.
- **Filename Reference** of the Program's Source Code.
- **Assigned Planned Output**. If a Planned Output of the assigned Program instance is assigned to the Report Set Entry, the system displays its name.
- **Definition Checked Out By** displays the username of the person who has checked out the Program definition, if it is checked out.

Creating and Setting Report Set Parameters

Report Sets have two sets of predefined Parameters—Overlay Template Parameters and Post-Processing Parameters—that determine how Oracle XML Publishes post-processes the Report Set, using templates, to generate a single- or multi-volume PDF output. The Parameters that are part of the Programs in a Report Set are used to execute the Programs during Report Set execution. You can define Parameters directly at the Report Set or Report Set Entry and share their values to Program Parameters to simplify Report Set submission.

When you create an Execution Setup for a Report Set or Report Set Entry, the system copies all these Parameters into the Execution Setup. You can modify them in the Execution Setup without affecting the Parameters in the Report Set or Program definitions.

This section contains the following topics:

- Setting Overlay Template Parameter Values
- Setting Post-Processing Parameter Values
- Setting Program Parameter Values
- Creating Parameters for Sharing Values within the Report Set

Setting Overlay Template Parameter Values

Oracle LSH Overlay Template definitions (OTDs) are designed to include all the template files you need to postprocess a Report Set; see Creating Overlay Templates. If you define your OTDs that way and this particular Report Set has no unusual characteristics, you can simply enter the name of the OTD in the Default OTD Parameter at the Report Set level and that value is propagated to all other OTD Parameters throughout the Report Set.

> **Note:**
>
> The value is displayed only where it is explicitly set.

However, if you need to use a different OTD for certain Report Set Entries or for certain Narratives or report outputs, you can do so. See Setting Overlay Template Parameters.

This section contains the following topics:

- Overlay Template Parameters
- Setting Overlay Template Parameters
- OTD Parameter Value Sharing

Overlay Template Parameters

The Report Set itself and Report Set Entries have a similar set of Overlay Template Parameters:

The following Parameters apply to the Report Set as a whole only:

- **Coversheet OTD** Oracle XML Publisher applies the specified OTD to the Report Set's coversheet.
- **TOC (Table of Contents) OTD** Oracle XML Publisher applies the specified OTD to the Report Set's table of contents.

The following Parameters apply to each Report Set Entry and the Report Set itself:

- **Default OTD**. If the value for any other OTD Parameter at the top Report Set level is null, XML Publisher uses the Default OTD value for that Parameter.
- **Pre-Narrative OTD**. Oracle XML Publisher applies the specified OTD to the Report Set Entry's Pre Narrative (text displayed on a page before the report), if any.
- **Content OTD**. Oracle XML Publisher applies the specified OTD to the Report Set Entry's generated report output, if any.
- **Post-Narrative OTD**. Oracle XML Publisher applies the specified OTD to the Report Set Entry's Post Narrative (text displayed on a page after the report), if any.

Setting Overlay Template Parameters

Normally you need to set a value only for the Default OTD Parameter at the Report Set level. Its value is automatically propagated to all other OTD Parameters unless you explicitly specify a different value. OTDs are designed to accommodate all possible needs in a Report Set; see Creating Overlay Templates.

To set an explicit value for an Overlay Template Parameter, do the following:

> **Note:**
>
> You must create at least one Overlay Template definition (OTD) before you can enter a value for the Default OTD Parameter. See Creating Overlay Templates for information.

1. In the **Default Value** column for an Overlay Template Parameter, click the plus (**+**) sign. A Search and Select screen opens.
2. Select the name of the Domain that contains the Overlay Template definition (OTD) that you want to use.
3. If the OTD is not contained directly in that Domain, drill down to the subdomain or Application Area that contains the OTD.
4. If you want to search for a specific OTD name or version label, select either Name or Version Label from the **Search By** drop-down list and enter the value.
5. Click **Go**. The system displays the OTDs in the location you specified. If you selected a name or version label it displays only OTDs with that name or version label.
6. Select the OTD you want.
7. Click **Apply**.

> **Note:**
>
> When you enter a value for an OTD Parameter, all the OTD Parameters of the same name below it in the Report Set hierarchy inherit that value unless you explicitly set a different value. Therefore if, for example, you are using one OTD throughout the Report Set with a few exceptions, you must set the Parameters of the same name in the immediate child Report Set Entries to the default OTD; see OTD Parameter Value Sharing.

OTD Parameter Value Sharing

OTD Parameters are predefined with automatic value sharing by name. That is, by default each OTD Parameter gets its value from the next higher OTD Parameter in the Report Set hierarchy with the same name. In addition, if there is no OTD Parameter with the same name higher in the Report Set hierarchy that has an explicitly defined value, it gets its value from the Default OTD Parameter at its own level. (The Default OTD Parameter for each Report Set Entry gets its value using automatic sharing by name from the next Default OTD Parameter above it in the Report Set hierarchy that has an explicitly defined value.)

If you do not want to use the default shared value you can explicitly set any OTD Parameter in any Report Set Entry. The corresponding Parameter of each of its child (and grandchild, and so on) Report Set Entries (whether existing or subsequently defined) gets the new value. If you want the child and grandchild Report Set Entries to use the old value, you must change the value of the corresponding Parameter for each of the immediate children of the Report Set Entry you changed.

All Overlay Template Parameters must have a value or Report Set postprocessing will not work. Therefore at the top level of the Report Set you must do one of the following if you want to use postprocessing:

- Specify a value for the Default OTD Parameter only. In this case, the other OTD Parameters at the Report Set level get their value from the Default OTD Parameter.
- Specify a value for the Default OTD Parameter and, if you want any of the other OTD Parameters to have a different value, specify a value for those Parameters.
- Specify a value for every OTD Parameter except the Default OTD Parameter.

Setting Post-Processing Parameter Values

Oracle XML Publisher uses the values of a predefined set of Post-Processing Parameters to control various aspects of generating a unified PDF output for the Report Set. There is a similar set of Parameters at the Report Set level and at the Report Set Entry level. Report Set Entry Post-Processing Parameters receive their default value from the corresponding Parameter of their immediate parent Report Set or Report Set Entry.

If you set these Parameter values at the Report Set level, the values you set become the default settings for the corresponding Parameters in the Report Set Entries below. For the Post-Processing Parameters that exist only in Report Set Entries, you can set the values in the top-level Report Set Entries for the same effect.

You can override the value for any Post-Processing Parameter for any Report Set Entry at any time. Changing a value for a Report Set Entry also changes the value of the same Parameter for any existing and future child Report Set Entries of that Report Set Entry.

This section contains the following topics:

- Post-Processing Parameters at the Report Set Level Only
- Post-Processing Parameters at the Report Set and Report Set Entry Levels

Post-Processing Parameters at the Report Set Level Only

The following Post-Processing Parameters appear at the Report Set level only:

- **Post Process:** If set to **Yes**, the Report Set uses Oracle XML Publishing postprocessing to apply overlay templates and generate one or more volumes of concatenated reports in PDF format.

  If set to **No**, the system disregards all other settings in the Post-Processing and Overlay Templates tabs. The system generates a table of contents and hyperlinks to every report, each of which is a separate file with the type specified for the corresponding Planned Output. This mode requires less time for execution and may be useful during development.
- **Single Volume Output Name:** If you enter a value here, the system ignores all Volume Breaks and produces a single postprocessed output, using this value as both the output name and title. The system also uses the value for the output filename, with the extension .pdf.
- **Start Page Number:** Enter a number (normally 1). If your Report Set is really a continuation of another Report Set that you have defined separately, you can enter

the number of pages in the first Report Set (plus 1) as the value for the Starting Page Parameter, to create continuous page numbering for both Report Sets.

Oracle XML Publisher can display page numbers as "1 of *x*," "2 of *x*," and so on. If you enter a Starting Page value of 400, Oracle XML Publisher displays page one as "400 of (*x*+400)" where *x* is the number of pages in the current Report Set.

- **Coversheet:** If set to **Yes**, Oracle XML Publisher incorporates the coversheet in the PDF output. The default setting is **Yes**.
- **Coversheet Orientation:** Select either Portrait (vertical) or Landscape (horizontal) for the orientation default value for the coversheet Overlay Template Definition. The default value is Portrait. See Creating Overlay Templates for more information.
- **TOC:** If set to **Yes**, Oracle XML Publisher generates a table of contents (TOC) for the Report Set. If set to **No**, Oracle XML Publisher does not generate a table of contents.

> **Note:**
>
> If TOC is set to **Yes** for a Report Set that contains multiple volumes, the system generates a single TOC in a separate PDF file.

- **TOC Orientation:** Select either Portrait (vertical) or Landscape (horizontal) for the orientation default value for the table of contents Overlay Template Definition. The default value is Portrait. See Creating Overlay Templates for more information.

Post-Processing Parameters at the Report Set and Report Set Entry Levels

The following Post-Processing Parameters appear at the Report Set Entry level and, unless otherwise noted, at the Report Set level:

- **Paper Size:** Select either A4 (European and Japanese standard) or US Letter (North American standard).

> **Note:**
>
> The system uses this setting to determine which OTD Files to use from the specified OTD. Setting this Parameter does not guarantee that your reports will print correctly for the size of paper. That depends on the OTD file itself and on the output of the Program.

- **Language:** In the Oracle LSH Release 2.5 there is only one option: English.
- **Calculated Title:** (Report Set Entry level only) Use this Parameter to receive a generated title for the Report Set Entry from a Program output Parameter whose value is propagated to `calculated_title`. The default value is null. If this value is null at runtime, the system uses the Report Set Entry's defined Full Title as the displayed title. See Passing Report Set Entry Values to and from Programs.
- **Shared Title:** (Report Set Entry level only) Use the `shared_title` Parameter to pass the Title properties values to the Program instance, so that the Program

instance can generate a different calculated title (the Program Title). See Setting Title Properties and Passing Report Set Entry Values to and from Programs.

- **Watermark Text:** Enter the text, if any, that you want in the watermark. (A watermark is text printed across the page so that it appears to be beneath the content of the report.) If the Overlay Template Definition file you specify does not have a watermark defined, the system ignores this value.

  If you want the watermark to display the current validation status (Development, Quality Control, or Production) of the Report Set or Report Set Entry, enter: `<VS>`

  To substitute your own terminology for any of the three validation statuses, enter three pipes (|) after the VS, each followed by the value you want to display when the validation status is Development, Quality Control, or Production, respectively. For example:

  – To display "Dev," "QC," or "Final," enter: `<VS|Dev|QC|Final>`

  – To display no watermark if the validation status is Production, but "Dev" or "QC" if the validation status is Development or Quality Control, enter: <VS|Dev|QC|>

  You can also have a string of fixed text on either side of the validation status or its equivalent, or only fixed text.

  – To display *y* text before the validation status and *x* text after it, enter: `y <VS|Dev|QC|Final> x`

  – To display the string xyz by itself, enter: `'xyz'`

Table 9-2 Examples of Validation Status-Based Watermark Values

Parameter Value	Development	Quality Control	Production
<VS>	Development	Quality Control	Production
<VS|Dev|QC|Final>	Dev	QC	Final
<VS|Dev|QC|>	Dev	QC	<none>
y < VS|Dev|QC|Final> x	y Dev x	y QC x	y Final x

- **Pre-Narrative Orientation:** Select either Portrait (vertical) or Landscape (horizontal) for the orientation default value for Overlay Template Definitions for narratives that appear before or instead of reports. The default value is Portrait. See Creating Overlay Templates for more information.
- **Post-Narrative Orientation:** Select either Portrait (vertical) or Landscape (horizontal) for the orientation default value for Overlay Template Definitions for narratives that appear after reports. The default value is Portrait. See Creating Overlay Templates for more information.
- **Hyperlink1...5 Target:** (Report Set Entry level only) You can set up your Overlay Template to display hyperlinks to other Report Set Entries. If your Overlay Template is set up for this, and you want to create a hyperlink to another Report Set Entry from the current one, click the Search icon and select its Full Title from the list of values.

  You can set up the Overlay Template to display up to five (5) of these hyperlinks to other Report Set Entries, using the subsequently numbered Hlink*x* Target Parameters.

- **Hyperlink1...5 Text:** (Report Set Entry level only) Enter the text to be displayed for each hyperlink. The default text is the Full Title of the Report Set Entry you selected for the corresponding HLink*x* Target Parameter.

Setting Program Parameter Values

When you submit the Execution Setup for a Report Set or a Report Set Entry, the system executes the associated Program(s) with the Parameter values supplied in the Execution Setup. You can set Program Parameter values and attributes in the Program definition or the Execution Setup definition.

Creating Parameters for Sharing Values within the Report Set

You can define your own Parameters at the Report Set level (or at the Report Set Entry level) for the purpose of sharing a single value to similar Parameters in Program instances throughout the Report Set (or in a particular Report Set Entry and its child Report Set Entries), and set up automatic value propagation from those Parameters.

For example, if the Report Set should run on data for a single study, and many Program instances included in the Report Set have a Parameter for Study Name or Study ID, define a Report Set-level Parameter Study Name and/or Study ID and set up automatic value propagation. Then the person submitting the Report Set for execution only needs to enter the study name or ID a single time.

See Setting Up Parameter Value Propagation.

> **Note:**
>
> Do not use spaces in the name of any Parameter you create for use in a Report Set. This will cause an error in postprocessing because the Parameter name becomes an HTML tag, where spaces are not allowed.

If a Parameter value should be the same throughout the Report Set, define the Parameter at the Report Set level.

If the Parameter value should be the same in only a single chapter of the Report Set, you can define the Parameter in the chapter-level Report Set Entry. If you set the Parameter to Required and the Report Set Entry Parameter does not have a value at runtime, the **Ready** flag is set to **No**, alerting the user to set a value. Alternatively, define the source Parameter at the top level the Report Set and name it in such a way that it is clear that it is used just in a particular Report Set Entry.

> **Note:**
>
> If a Program Parameter gets its value from another Parameter in a Report Set, at runtime the system gets the value from the source Parameter even if the source Parameter is not included in the submission (because it is in a Report Set Entry that is not included). You can then enter a value manually. If the source Parameter is required and does not have a value, you get an error message.

To create a Parameter for value sharing, do the following:

1. Go to the **Parameters** subtab of the Report Set or Report Set Entry.
2. Click **Add**.
3. Create a Parameter, following instructions in Defining Parameters .
4. If you want to pass the value of this Parameter to all others in the Report Set (or Report Set Entry) with the same name, do the following:
    a. In the Parameter Instance screen, click **Update**.
    b. Under Value Propagation, select **Automatically pass value(s) to parameters with the same name**.
    c. Click **Apply**.

When you add a Program instance to the Report Set that should receive a Parameter value from this Parameter, define value propagation from this Parameter for the Program instance Parameter; see Setting Up Value Propagation from the Target Parameter.

Defining Report Set Entries

Each Report Set Entry can contain one or more of the following:

- Other Report Set Entries—these are subsections, or children, of the Report Set Entry within which they are defined
- One assigned Program instance and Planned Output to generate a report in the chapter or section represented by the Report Set Entry.
- Pre-Narrative—a text narrative displayed either before a report or instead of a report in the chapter or section represented by the Report Set Entry. You can either enter text or upload text from a file
- Post-Narrative—a text narrative displayed after a report in the chapter or section represented by the Report Set Entry. You can either enter text or upload text from a file.

You assign a Program instance and Planned Output to a Report Set Entry from the Report Set Structure view (see Assign Planned Output) or from a Report Set Entry's Properties screen (see Assigning a Planned Output to a Report Set Entry).

This section contains the following topics:

- Creating Multiple Report Set Entries
- Setting Report Set Entry Properties
- Adding Narratives

Creating Multiple Report Set Entries

To create a Report Set Entry, navigate to the Report Set or Report Set Entry under which you want to create a new Report Set Entry, and:

1. Click **Add**. The Add Entries screen appears.
2. Enter values as necessary in the following fields for each Report Set Entry you want to add at this level. Click **Add 30 Entries** if you need any additional rows.

- **Entry No. Prefix**. (Optional) The system uses text you enter here, if any, in the Full Title in the table of contents to identify the Report Set Entry. You can use the prefix and suffix to include information that is not part of the actual title of the Report Set Entry but which you need to display. For example, you may require a set of reports so large that you decide to create multiple Report Set definitions to manage it, with one for each volume. To make the numbering unique across the larger set of reports, you must add the volume number as a prefix to each Report Set Entry number.

  Oracle truncates spaces at the beginning and end of text in fields. If you want a space to appear between the prefix and the parent number, type `\sp\` instead of using the space bar on the keyboard.

  If you want a delimiter character to appear between the prefix and the parent number, you must type it here.

  If you want the system to include the Entry No. Prefix in the Program Title that is available to pass to the Program assigned to the Report Set Entry, leave the checkbox selected. If not, deselect it. See Passing Report Set Entry Values to and from Programs for further information.

- **Parent Number**. This read-only field contains the number of the Report Set Entry that is the parent of all the Report Set Entries listed here. (If these Report Set Entries are at the top level in the Report Set, this column contains no value.)

  The system automatically concatenates parent and child Report Set Entries' numbers. If parent Report Set Entry 10 contains child Report Set Entries 1-3, the system gives the child Report Set Entries numbers 10.1, 10.2, and 10.3. If Report Set Entry 10.2 has two child Report Set Entries, they become 10.2.1 and 10.2.2, and so on.

  You use the Delimiter Parameter to determine what punctuation the system inserts between numbers; see Delimiter below.

  If you want the system to include the Parent Number in the Program Title that is available to pass to the Program assigned to the Report Set Entry, leave the checkbox selected. If not, deselect it. See Passing Report Set Entry Values to and from Programs for further information.

- **Delimiter**. Enter the character you want the system to use to separate the Entry Number from the Entry Number of the parent Report Set Entry. If there is no parent Report Set Entry, this value has no effect. The default is a period (full stop) (.), which is used in the examples under Parent Number above. In those examples, if you entered a comma as the Delimiter, for example, the Report Set Entry numbers would be displayed as 10,1 10,2 and 10,3 and 10,2,1 and 10,2,2. Oracle truncates spaces at the beginning and end of text in fields. If you want a space to appear between the delimiter and entry number, type \sp\ instead of using the space bar on the keyboard.

  If you want the system to include the Delimiter in the Program Title that is available to pass to the Program assigned to the Report Set Entry, leave the checkbox selected. If not, deselect it. See Passing Report Set Entry Values to and from Programs for further information.

- **Entry No**. Accept the default value or enter the number that represents the order number of this Report Set Entry in relation to other Report Set Entries contained within the same parent Report Set Entry.

> **Note:**
>
> If you enter numbers in a nonsequential order, after you apply your changes, the system displays sibling Report Set Entries in order, using the numbers you entered, provided that the set of entry numbers does not violate the numbering settings defined in the parent Report Set or Report Set Entry: if **Strict** is set to Yes, numbers must be sequential, with no gaps; if **Unique** is set to Yes, numbers must be unique within the parent.
>
> To change the order, you can use the **Reorder** feature. See Reordering and Renumbering Objects for further information.

If you want the system to include the Entry No. in the Program Title that is available to pass to the Program assigned to the Report Set Entry, leave the checkbox selected. If not, deselect it. See Passing Report Set Entry Values to and from Programs for further information.

- **Entry No. Suffix**. (Optional) The system uses text you enter here in the Full Title in the table of contents to identify the Report Set Entry. If you need additional numbers and/or text to follow the Report Set Entry's number in its full title, enter it here. For example, you can include the word "Table," "Listing," or "Figure" (or any other text string) as the suffix.

  If you want a space to appear between the entry number and the suffix, type `\sp\` at the beginning of the suffix.

  If you want the system to include the Entry No. Suffix in the Program Title that is available to pass to the Program assigned to the Report Set Entry, leave the checkbox selected. If not, deselect it. See Passing Report Set Entry Values to and from Programs for further information.

- **Title**. Enter a title for the Report Set or Report Set Entry. The system displays this value as part of the Full Title in the table of contents to identify the Report Set Entry. The title value is not required to be unique in the Report Set. You can pass its value to the Program that generates the Report Set Entry's Planned Output. The system passes the title value to the Report Set Entry name, which is used internally only. (The internal name must be unique. If another Report Set Entry exists with the same name, the system creates a unique name for the newer Report Set Entry by appending or incrementing `_1`.)

  This field is required unless the Placeholder flag is set to Yes, in which case the system gives the Title the value: `Placeholder`.

  > **Note:**
  >
  > The system concatenates the values of the fields up to and including the Title, in the order shown, to create the *Full Title* for the Report Set Entry.

- **Subtitle** (Optional; Oracle LSH does not use this value). Enter a subtitle for the Report Set Entry. The system can pass this value as a Parameter value for

post-processing; see Passing Report Set Entry Values to and from Programs for further information.

- **Report Type**. (Optional; Oracle LSH does not use this value). From the drop-down list, select Figure, Listing, or Table. The system can pass this value as a Parameter value during execution; see Passing Report Set Entry Values to and from Programs for further information. The values displayed are configurable. Instructions are in the *Oracle Life Sciences Data Hub System Administrator's Guide*
- **Placeholder**. Set to **Yes** to exclude the Report Set Entry from the table of contents. You can use this feature to force the system to accept unconventional numbering. The system does not display a placeholder Report Set Entry in the generated table of contents.

  For example, if you have chapter 16 and subsections 16.1.1 and 16.1.2 but no section 16.1, define section 16.1 as a placeholder. In the table of contents, the system does not display 16.1 but it does display chapter 16 and subsections 16.1.1 and 16.1.2.

  The default value is **No** (the report set entry is not a placeholder).
- **Strict**. Set to **Yes** if you want to force numbering of Report Set Entries within the Report Set Entry to be sequential, with no gaps. Set to **No** if you want to allow gaps.
- **Unique**. Set to **Yes** if you want to enforce unique numbering of Report Set Entries within the Report Set Entry. Set to **No** if you want to allow duplicate numbers.
- **Volume Break**. Set to **Yes** if you want this Report Set Entry to start a new volume in the post-processed PDF Report Set output.
- **Volume Name**. If **Volume Break** is set to **Yes**, enter the title you want to give the volume that will begin with this Report Set Entry. If **Volume Break** is set to **No**, the system ignores the **Volume Name**, if any.

3. Repeat for as many Report Set Entries as you need with the same parent Report Set or Report Set Entry.
4. Click **Apply**. The Report Set instance screen appears. To complete the definition of these Report Set Entries, navigate to each Report Set Entry in turn and do the following:
   - (Optional) Click **Assign** to assign a Program instance and Planned Output to the Report Set Entry. This is required if you want to include a report in the chapter or section represented by the Report Set Entry; but it is optional because you can have Report Set Entries that serve only to organize their subentries and/or display narrative text. See Assigning a Planned Output to a Report Set Entry.
   - (Optional) Click **Add** a **Pre-Narrative** or **Post-Narrative**. Pre- and post-narratives are text that appears immediately before or after the actual generated report, or instead of a report, in a Report Set Entry. See Adding Narratives.
   - (Optional) Click **Update**. The system makes all other Report Set Entry property fields modifiable; see Setting Report Set Entry Properties.
   - (Optional) Create Report Set Entry Parameters and/or change the default value of Overlay Template and Post-Processing Parameters; see Creating and Setting Report Set Parameters.

Setting Report Set Entry Properties

In the main Report Set Entry screen you can set values for many Report Set Entry properties. Reach this screen by clicking on the Report Set Entry's name in the Report Set Structure view, or by navigating to it from the Report Set's Properties screen.

To modify most Report Set Entry properties you must click **Update**. However, to assign a Program instance and/or Planned Output or to add a Narrative, you must *not* click **Update**.

This section contains the following topics:

- Setting Overall Properties
- Setting Title Properties
- Setting Assigned Program Properties
- Adding Narratives
- Setting Numbering Properties
- Setting Volume Break Properties

Setting Overall Properties

Enter values for the following properties for the Report Set Entry:

- **Title:** Enter a title for the Report Set or Report Set Entry. The system displays this value as part of the Full Title in the table of contents to identify the Report Set Entry. The title value is not required to be unique in the Report Set. The system passes the title value to the Report Set Entry name, which is used internally only and is required to be unique. If another Report Set Entry exists with the same name, the system creates a unique name for the newer Report Set Entry.

  This field is required unless the Placeholder flag is set to Yes.
- **Subtitle:** (Optional; Oracle LSH does not use this value.) Enter a subtitle for the Report Set or Report Set Entry. The system can pass this value as a Parameter value for post-processing; see Passing Report Set Entry Values to and from Programs for further information.
- **Description:** (Optional) Enter a description for the Report Set Entry.
- **Report Type:** (Optional; Oracle LSH does not use this value.) Select the expected output type; for example, Table, Figure, or Listing. The system can pass this value as a Parameter value during execution; see Passing Report Set Entry Values to and from Programs for further information. The values displayed are configurable. Instructions are in the "Modifiable Lookups" section of the lookups chapter in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

> **Note:**
>
> To change the Program Title or Full Title, see Setting Title Properties.

Setting Title Properties

The system concatenates the title properties (except Placeholder) in the order shown below, plus the Title, to generate the Full Title. The Report Set table of contents displays the Full Title.

Select the checkbox of each property you want the system to concatenate with the Title to generate the Program Title. The system can pass the Program Title to the assigned Program instance during execution; see Passing Report Set Entry Values to and from Programs for further information.

For information about each property, see Creating Multiple Report Set Entries.

Setting Assigned Program Properties

Report Set Entries have format-related properties whose values can be passed to the assigned Program instance during Report Set execution. In your Program source code you can use these values to format the report output; see Passing Report Set Entry Values to and from Programs. Oracle LSH does not use these values. The following properties are available for use. You can add allowed values for some of them. Instructions are in the chapter "Adding and Modifying Lookup Values" in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

A Report Set Entry inherits default values for the Assigned Program Properties from its parent Report Set Entry. The top-level Report Set uses system defaults. You can reset them manually at any level.

> **Note:**
>
> You can assign a Program instance and Planned Output to a Report Set Entry in either the Report Set Structure View or in the Properties screen of the Report Set Entry.

- **Page Size Horizontal:** (Number) Enter the horizontal measurement of the page. If your company does not configure a list of values, the default value is 210 (cms).
- **Page Size Vertical:** (Number) Enter the vertical measurement of the page. If your company does not configure a list of values, the default value is 297 (cms).
- **Page Size Unit:** The unit in which the horizontal and vertical measurements are given. If your company does not configure a list of values, the default value is cms (centimeters).
- **Program Title Font Name:** Select the name of the font in which you want the title to appear. If your company does not configure a list of values, the default value is Courier.
- **Program Title Font Size:** (Number) Enter the size of the font in which you want the title to appear. If your company does not configure a list of values, the default value is 10.
- **Footer:** Enter the text you want to use as the footer on each page of the report, up to 200 characters.

- **Page Size Margin Left:** (Number) Enter the size of the margin you want at the left side of the page. If your company does not configure a list of values, the default value is 3.42 (cms).
- **Page Size Margin Top:** (Number) Enter the size of the margin you want at the top of the page. If your company does not configure a list of values, the default value is 3.42 (cms).

Adding Narratives

You can add and remove Narratives only when you are not in update mode; see Adding Narratives.

Setting Numbering Properties

You can modify the numbering properties as follows:

Strict: If you want to require that all Report Set Entries contained in this Report Set Entry have sequential numbering, with no gaps, select **Yes**. The default value is shared from the parent Report Set Entry or Report Set.

Unique: If you want to require that each Report Set Entry contained in this Report Set Entry have a unique number within this Report Set Entry, select **Yes**. The default value is shared from the parent Report Set Entry or Report Set.

Setting Volume Break Properties

Set the Volume Break properties as follows:

Volume Break: If you want this Report Set Entry to be the first in a separate PDF file, or volume of the Report Set, select **Yes**. The default value is **No**.

Volume Name: If you want this Report Set entry to be the first in a separate PDF file, or volume of the Report Set, enter the name of the volume you want it to start.

> **Note:**
>
> If you enter a Volume Name but set Volume Break to **No**, Oracle XML Publisher does not create a volume break at this Report Set Entry (and does not use the Volume Name value).
>
> You can override these settings at the Report Set level by defining a value for the Report Set Post-Processing Parameter Single Volume Output Name. If that Parameter has a value, the system generates a single volume with the name specified and ignores all Report Set Entry volume-related values.

Adding Narratives

A narrative is text that you can add to a Report Set Entry either in addition to or instead of reports generated by a Program instance. If a Report Set Entry contains both narratives and a Program instance, the narratives appear on the same page as reports either immediately before or after the report generated by the Program instance.

Narratives added as Pre-Narratives appear before the report, and narratives added as Post-Narratives appear after the report.

To add a narrative, you must not be in Update mode.

1. In the Report Set Entry's Properties screen, click either the Pre-Narrative or Post-Narrative **Add** button. The appropriate Add screen opens.
2. Either upload a file containing the text or enter the text:
   - To upload text, click **Browse**. The Choose file window opens. Browse and select the text file. Click **Open**. The system copies the text into the Narrative Description text box.

     > **Note:**
     >
     > The system does not store the file in Oracle LSH when you upload it. The system copies the text content as a string.

   - Alternatively, enter text in the Narrative Description text box.

     > **Note:**
     >
     > If you both upload a text file and enter text, the file you upload overwrites the text you enter.

3. Click **Apply**. The system returns you to the Report Set Entry screen.

Defining Programs to Generate Reports

This section contains the following topics:

- About Programs in Report Sets
- Assigning a Planned Output to a Report Set Entry
- Viewing Planned Output Assignments
- Passing Report Set Entry Values to and from Programs

About Programs in Report Sets

A Report Set's reports are generated by Program instances in the Report Set. You can create a Program definition and instance from within a Report Set, or you can create an instance in a Report Set of a Program definition in an Application Area or Domain. You can have Programs of different technology types (for example, SAS, PL/SQL, and Oracle Reports) in the same Report Set. See Defining Programs for general information about Programs in Oracle LSH.

Each Program instance in a Report Set can have any number of primary Planned Outputs, each of which corresponds to a single report output. You can assign each Planned Output to a Report Set Entry. When you execute the Report Set, each report appears in the Report Set in the location defined by its Report Set Entry.

Each Program instance in a Report Set is executed once during Report Set execution and can be submitted with only one set of Parameter values. Therefore, when you change a Program Parameter value in the Execution Setup definition or at submission under one Report Set Entry, the value is effectively changed for all Report Set Entries assigned to the same Program instance.

Similarly, when you execute the Report Set, you must select a single currency and blinding status for the Report Set instance.

Only one person can check out and work on a particular Program instance in a Report Set at a time. In addition:

- Programs that are checked out by other people when you execute a Report Set are not executed and a message to that effect appears in the Report Set output if you create an Overlay Template for that purpose (see Creating an Overlay Template Definition). The information is also in the Report Set's log file. If you have a Program checked out, the system executes the installed version of it.
- Programs that are checked out during Report Set installation by another user are not installed; see Installing Report Sets.

Developing Programs inside Report Sets is different from developing independent Programs in the following ways:

- **Planned Outputs**. You can assign each primary Planned Output that you want to include in the Report Set to a different Report Set Entry. Do this in the Report Set Entry or the Report Set Structure view; see Assigning a Planned Output to a Report Set Entry. You are not required to assign every Planned Output to a Report Set Entry. Any unassigned Planned Outputs are not included in the Report Set output.
- **Mapping**. All the standard Oracle LSH methods of mapping are available:
  - **Automatic Mapping by Name** is available for multiple Program instances at once from the Report Set Structure View and for one Program instance at a time from the Program View.
  - **Create Table Descriptors from Existing Table Instances**, which includes mapping, is available from the **Actions** drop-down on each Program instance's Properties screen.
  - **Create Table Instances from Existing Table Descriptors**, which includes mapping, is available from the **Actions** drop-down on the Report Set's Properties screen. You can select Table Descriptors from all Program instances in the Report Set.

    > **Note:**
    >
    > If you map or modify the mappings of a checked-in Program instance, the system implicitly checks out the Program instance. This ensures that the Program instance and associated mappings must be installed (and therefore versioned) if any of its mappings are modified.

- **Parameters**. As with Programs outside of Report Sets, you can define Parameters as necessary and set them in the Program instance, in the Execution Setup, or at runtime. In addition, you can pass the values of Report Set Entry properties to the

Program instance and back; see Passing Report Set Entry Values to and from Programs.

You can also define Parameters directly at the Report Set or Report Set Entry level for the purpose of sharing their value with Program input Parameters at runtime, and you can populate the value of one Program's input Parameter with the value of another Program's output Parameter at runtime; see Creating Parameters for Sharing Values within the Report Set.

- **Output Reuse**. Unless you specify otherwise, Oracle LSH does not regenerate an output if the Program instance version, data currency, and Parameter values have not changed since the last execution that produced a nonretired output; see Output Reuse .

  You can force the system to run the job and regenerate the output by setting the system Parameter **Force Execution** to **Yes** in the Execution Setup or at submission.

> **Note:**
>
> Setting **Force Execution** to **Yes** reruns all the Program instances whose Report Set Entries are included in the execution. Use this option only if you want all of these outputs to be reproduced.

> **Note:**
>
> It is possible to include Program instances in a Report Set that transform data, or that both transform and report data. However, for the best Report Set execution performance, Oracle recommends including only Program instances that report data, not Programs that transform data, inside a Report Set. Instead, run transform Program instances outside the Report Set and use their target Table instances as the source Table instances for Program instances in the Report Set.

You can add Programs, map their Table Descriptors, and submit them for execution in several different Oracle LSH screens. See the following sections for details:

- **Report Set Structure View**. To reach the Structure view, click the Report Set's hyperlink under the Work Area and then select Structure from the View drop-down if it is not already selected; see Building and Modifying the Report Set.
- **Report Set Program View**. To reach the Structure view, click the Report Set's hyperlink under the Work Area and then select Program from the View drop-down if it is not already selected; see Options from the Report Set Program View Screen
- **Report Set Entry Add Screen**. To reach the Add Report Set Entries screen, go first to the Report Set Properties screen by clicking its hyperlink in the Report Set Structure view. Then click **Add Entries**; see Creating Multiple Report Set Entries.
- **Report Set Entry Properties Screen**. To reach a Report Set Entry's Properties screen, click its hyperlink in the Report Set Structure view; see Setting Report Set Entry Properties.

Assigning a Planned Output to a Report Set Entry

After you assign a Program instance to a Report Set Entry, you can assign its primary Planned Outputs to other Report Set Entries in the same Report Set. The system then places the actual report generated by the Program instance for that Planned Output in the chapter or subsection corresponding to the Report Set Entry.

If you select a Report Set Entry that already has a Planned Output assigned, you can change the assignment by selecting a different one.

You can assign a Planned Output as follows:

- In the Report Set Structure view, select the Report Set Entry to which you want to assign a Planned Output, select **Assign Planned Output** from the drop-down list, and click **Go**.
- In the Report Set Entry Properties screen (not in Update mode), click the **Assign** button.

In both cases, the Assign Program Instance and Planned Output screen opens. Do the following:

> **Note:**
>
> If the Report Set Entry already has a Planned Output assigned, the screen is populated with the Planned Output's Program instance and all its Planned Outputs. You can select a different Planned Output of the same Program or follow the instructions below to select a different Program instance and Planned Output.

1. In the **Search By Program Instance** field, click the Search icon. The Search and Select window opens.
2. Click **Go**, or enter the name of the Program instance whose Planned Output you want to assign and then click **Go**.

   The system lists Program instances already assigned to the Report Set. If you have entered the name of the Program instance, the system displays only that Program instance.

   > **Note:**
   >
   > The system displays only checked-in Program instances.

3. Click the icon in the Quick Select column for the Program instance whose Planned Output you want to assign.

   The system returns you to the Assign Planned Output screen with the Program instance name displayed.
4. Click **Go**. The system displays the Planned Outputs of the selected Program instance.

If a Planned Output is already assigned to a Report Set Entry, the system displays that Report Set Entry's number and title in the **Assigned to RSE** column.

5. Select the Planned Output you want to assign and click the **Select** button. The system assigns the Planned Output to the current Report Set Entry. If you select a Planned Output that is already assigned to another Report Set Entry, the system removes it from that Report Set Entry.

- Options from the Report Set Program View Screen

Options from the Report Set Program View Screen

To reach the Program View screen, navigate to the Report Set in the Applications tab hierarchy and click its hyperlink. Then select **Program** from the **View** drop-down list.

In the Report Set Structure view you can build and modify the Report Set structure and do many other things as follows:

1. Select a Program instance.
2. Select one of the following items from the drop-down list.
3. Click **Go**.

The following actions are available:

- **Copy**. The system opens the Paste screen and copies the Program instance you selected into the location you specify: either a Report Set or a Work Area. The system copies the Program instance's Execution Setup(s) and Table Descriptor mappings. See Copying, Cloning, and Moving Objects for instructions.
- **Default Execution Setup**. If a default Execution Setup already exists for this Program, the system displays it. If a default Execution Setup does not yet exist, the system automatically creates one; see Creating, Modifying, and Submitting Execution Setups for further information.
- **Install**. The system installs the selected Program instance. You can install a Program instance if it is checked in or if you have checked it out.
- **Map**. The standard Mapping screen opens. The Table Descriptors of this Program only are available for mapping; see Defining and Mapping Table Descriptors. You can map Table Descriptors for multiple Program instances at once in the Report Set Structure view.
- **Remove**. The system deletes the selected Program instance. If the Program instance is assigned to one or more Report Set Entries, you get a warning. You can see where its Planned Outputs are assigned by clicking its hyperlink to open its Properties screen and then clicking the Planned Outputs tab.

  The Program definition is not affected.
- **Quick Submit**. The system submits the Program instance using the default Execution Setup with its default values, without opening the Execution Setup screen. The system upgrades the Execution Setup if necessary and validates it. The submission fails if:
  - The Execution Setup cannot be upgraded; for example, if another user is currently modifying the default Execution Setup
  - The Execution Setup is invalid; for example, the Parameters that apply to the portion of the Report Set being submitted have invalid values

In the Job Execution section of the My Home page you see two jobs. A temporary job starts the actual report generation job.

- **Submit**. The system opens the submission screen for the default Execution Setup. From there you can set Parameters and submit the Program instance. The system automatically includes in the submission all Report Set Entries to which the Program is assigned. See Creating, Modifying, and Submitting Execution Setups.

  To submit a different Execution Setup, click the Program's hyperlink to open its Properties screen. Select **Execution Setups** from the **Actions** drop-down list and follow instructions in Creating, Modifying, and Submitting Execution Setups.

> **Note:**
>
> You can submit a Program instance in a Report Set only if it is installed and assigned to at least one Report Set Entry.

Example 9-1 View Programs

Click a Program's hyperlink in the Name column to go to that Program's Properties screen.

Viewing Planned Output Assignments

You can see information about Planned Output assignments in the following places:

- In the Report Set instance's Program view, there is a column that displays the number of primary Planned Outputs each Program has, and another column for the number of primary Planned Outputs that are not currently assigned to a Report Set Entry. To see which Planned Outputs are assigned to which Report Set Entries, click the hyperlink on the Program's name, then open the Planned Outputs tab.
- In the Report Set instance's Structure view, there is a column for Assigned Planned Output that displays the name of the Planned Output assigned to each Report Set Entry.
- In the Properties screen for each Report Set Entry, the system displays the name of the Program instance and Planned Output currently assigned to the Report Set Entry. This is the only place where you can create and remove Planned Output assignments. See Assigning a Planned Output to a Report Set Entry for instructions.

Passing Report Set Entry Values to and from Programs

You can pass the values of title- and formatting-related Report Set Entry properties to the Program assigned to a Report Set Entry, use these values in the Program's source code during execution, and then send a values back to the Report Set for use during postprocessing.

Different Report Set Entries assigned to the same Program instance can have different values for the same Report Set Entry properties. If you use this feature, your source code must handle the input from different Report Set Entries, and the output to different Report Set Entries, correctly.

This section contains the following topics:

- Report Set Entry Properties Available for Passing
- Passing Values from a Report Set Entry to a Program Instance
- Passing Values from a Program Instance to the Report Set for Post-Processing

Report Set Entry Properties Available for Passing

You can pass the values of Report Set Entry assigned program properties and several additional properties from a Report Set Entry to its assigned Program and use them in the Program to format the output of the Report Set Entry. See Setting Assigned Program Properties for a description of each Program property.

You can also pass values for the following:

- **File Reference Name** Each Planned Output can have only one file reference name, and each Planned Output is assigned to only one Report Set Entry. Oracle recommends using the file reference name in your Program code to distinguish the properties of one Report Set Entry from another one to which another Planned Output of the same Program instance is assigned. However, Oracle LSH does not enforce unique File Reference Names among a Program's Planned Outputs; you must do that manually.
- **Full Title**. The system uses the full title in the Report Set's table of contents. The Full Title includes all of the following, concatenated in the following order:

  Entry No. Prefix||Parent No.||Delimiter||Entry No.||Entry No. Suffix||" " || Title
- **Program Title**. The system does not use the Program title. It always includes the Title and can include any of the other elements that the Full Title can, in the same order, if you so specify; see Setting Title Properties and Passing Report Set Entry Values to and from Programs.

Passing Values from a Report Set Entry to a Program Instance

The method of passing Report Set Entry values to a Program instance differs depending on whether you are working in SAS or Oracle Reports or PL/SQL.

- SAS

SAS

Oracle LSH includes a SAS data set named LSH_RS.RS_TITLE that is available when you open the SAS IDE or execute a SAS Program from within the context of a Report Set. LSH_RS.RS_TITLE includes the values of all Program properties, plus the file reference name, Full Title, Program Title, and Report Type for all the Report Set Entries to which the Program instance is assigned. The primary key is the file reference name of the Planned Output assigned to each Report Set Entry.

> **Note:**
>
> Oracle LSH does not enforce file reference name uniqueness among a Program's Planned Outputs. If you plan to pass values from a Report Set Entry to its Program instance, you must ensure that each Planned Output assigned to a Report Set Entry has a different file reference name.

Your source code can reference any of the values in LSH_RS.

Table 9-3 Report Set Entry Properties and Corresponding LSH_RS Data Set Variables

Property	Number	Variable	Type	Length	Format	Informat	Label
File Reference Name	1	FILE_REF	Char	8	$8.	$8.	FILE_REF
Title	2	TITLE	Char	4000	$4000.	$4000.	TITLE
Full Title	3	TOCTITLE	Char	4000	$4000.	$4000.	TOCTITLE
Program Title	4	RSETITLE	Char	200	$200.	$200.	RSETITLE
Prefix	5	PREFIX	Char	200	$200.	$200.	PREFIX
Parent No.	6	PARENT	Char	4000	$4000.	$4000.	PARENT
Delimiter	7	DELIMIT	Char	30	$30.	$30.	DELIMIT
Entry No.	8	ENTRY	Char	200	$200.	$200.	ENTRY
Suffix	9	SUFFIX	Char	200	$200.	$200.	SUFFIX
Subtitle	10	TITLE2	Char	200	$200.	$200.	TITLE2
Report Type	11	REP_TYPE	Char	4000	$100.	$4000.	REP_TYPE
Footer	12	FOOTER	Char	200	$200.	$200.	FOOTER
Program Title Font Name	13	FONTNAME	Char	4000	$20.	$4000.	FONTNAME
Program Title Font Size	14	FONTSIZE	Num	8	6.	7.	FONTSIZE
Page Size Horizontal	15	PS_HORI	Num	8	6.	7.	PS_HORI
Page Size Vertical	16	PS_VERT	Num	8	6.	7.	PS_VERT
Page Size Unit	17	PS_UNIT	Char	20	$20.	$20.	PS_UNIT
Page Size Margin Left	18	PS_LEFT	Num	8	6.2	8.2	PS_LEFT
Page Size Margin Top	19	PS_TOP	Num	8	6.2	8.2	PS_TOP
Volume Name	20	VOL_NAME	Char	255	$255.	$255.	VOL_NAME

Passing Values from a Program Instance to the Report Set for Post-Processing

You call Oracle LSH APIs from either SAS or Oracle Reports or PL/SQL to pass values from the executed Program instance to the Report Set for postprocessing.

Refer the *Oracle Life Sciences Data Hub Application Programming Interface Guide* for details.

- SAS
- Oracle Reports or PL/SQL

SAS

The example function below uses APIs to retrieve values from the Program instance and pass them to Oracle LSH for Report Set postprocessing. It uses a PL/SQL wrapper that you call from SAS. Within the wrapper, the function calls several Oracle LSH public APIs to do the work.

If a particular Planned Output is not assigned to a Report Set Entry when you execute the Program instance, the function returns "Title for *fileref_name*" with the fileref name of the unassigned Planned Output.

Example 9-2 Function to Pass Title and Other Values from SAS to Oracle LSH

```
SAS code
--------------------------------------------------------------------------------
Proc SQL;
/*set the job context then send the output value*/
connect to oracle (user=%sysget(CDR_SCHEMA) pass=%sysget(CDR_PASSWD)
path=%sysget(CDR_DB) );

/* pass output parameter, sub title and title back to LSH */
execute(exec my_plsql_package.CallLSHApi(
'MyParamName'
,'My Param Value'
,'out1'
,'My Output Title'
,'out1'
,My Output Sub Title'
)
by oracle ;

PL/SQL code
--------------------------------------------------------------------------------
CREATE OR REPLACE PACKAGE my_plsql_package AS
Procedure CallLSHApi(
pi_vParamName IN varchar2
,pi_vParamValue IN varchar2
,pi_vTitleFileRef IN varchar2
,pi_vTitle IN varchar2
,pi_vSubTitleFileRef IN varchar2
,pi_vSubTitle IN varchar2
);
END my_plsql_package;
/

CREATE OR REPLACE PACKAGE BODY my_plsql_package AS
Procedure CallLSHApi(
pi_vParamName IN varchar2
,pi_vParamValue IN varchar2
,pi_vTitleFileRef IN varchar2
,pi_vTitle IN varchar2
,pi_vSubTitleFileRef IN varchar2
,pi_vSubTitle IN varchar2
) IS

return_status VARCHAR2(10);
msg_count NUMBER;
msg_data VARCHAR2(2000);
BEGIN
   Cdr_Pub_Exe_User_Utils.setOutputParams(p_api_version => 1
               ,p_init_msg_list => Cdr_Pub_Def_Constants.G_FALSE
               ,p_commit => Cdr_Pub_Def_Constants.G_FALSE
               ,p_validation_level => Cdr_Pub_Def_Constants.G_VALID_LEVEL_FULL
               ,x_return_status => return_status
               ,x_msg_count => msg_count
               ,x_msg_data => msg_data
               ,pi_vparamName => pi_vParamName
               ,pi_vparamValue => pi_vParamValue) ;
   IF return_status <> 'S' THEN
      RAISE_APPLICATION_ERROR(-20200,'Failed to call
Cdr_Pub_Exe_User_Utils.setOutputParams: '||msg_data);
   END IF ;
```

```
   Cdr_Pub_Exe_User_Utils.SetCustomOutputTitle(p_api_version => 1
               ,p_init_msg_list => Cdr_Pub_Def_Constants.G_FALSE
               ,p_commit => Cdr_Pub_Def_Constants.G_FALSE
               ,p_validation_level =>Cdr_Pub_Def_Constants.G_VALID_LEVEL_FULL
               ,x_return_status => return_status
               ,x_msg_count => msg_count
               ,x_msg_data => msg_data
               ,pi_vFileRef=> pi_vTitleFileRef
               ,pi_vValue => pi_vTitle ) ;
   IF return_status <> 'S' THEN
      RAISE_APPLICATION_ERROR(-20200,'Failed to call
Cdr_Pub_Exe_User_Utils.SetCustomOutputTitle: '||msg_data);
   END IF ;
   Cdr_Pub_Exe_User_Utils.SetCustomOutputSubTitle(p_api_version => 1
               ,p_init_msg_list => Cdr_Pub_Def_Constants.G_FALSE
               ,p_commit => Cdr_Pub_Def_Constants.G_FALSE
               ,p_validation_level => Cdr_Pub_Def_Constants.G_VALID_LEVEL_FULL
               ,x_return_status => return_status
               ,x_msg_count => msg_count
               ,x_msg_data => msg_data
               ,pi_vFileRef=> pi_vSubTitleFileRef
               ,pi_vValue => pi_vSubTitle ) ;
   IF return_status <> 'S' THEN
      RAISE_APPLICATION_ERROR(-20200,'Failed to call
Cdr_Pub_Exe_User_Utils.SetCustomOutputSubTitle: '||msg_data);
   END IF ;
END CallLSHApi;
END my_plsql_package;
/
```

Oracle Reports or PL/SQL

To pass Title and Program property values from a Program instance to the Report Set for postprocessing in a PL/SQL Program, you can call Oracle LSH public APIs:

- **Cdr_Pub_Exe_User_Utils.SetCustomOutputTitle**. Use this API to pass the title from the Report Set Entry to the Program instance.
- **Cdr_Pub_Exe_User_Utils.SetCustomOutputSubTitle**. Use this API to pass the subtitle from the Report Set Entry to the Program instance.
- **Cdr_Pub_Exe_User_Utils.setOutputParams**. Use this API to pass every other value you need from the Report Set Entry to the Program instance.

Installing Report Sets

Installing a Report Set is complicated because of its size and the fact that many people can work on it at the same time. Depending on the exact timing of your action and other people's actions, you may receive an error message that the system cannot perform your action immediately because of a conflict with another user. Your work will not be lost and you can try again in a short time.

At the end of the installation process the system upgrades all checked in Report Set Execution Setups as necessary to synchronize them with the Report Set definition.

> **Note:**
>
> If someone is working on an Execution Setup, the Execution Setup is implicitly checked out. Checked out Execution Setups are not upgraded during installation. No message is displayed.

During the installation process, the Report Set instance itself (which includes the mappings of its Program instances to Table instances) is unavailable for modification. If the Report Set is very large the installation process may take a long time, so plan your installation accordingly.

To see a list of reasons Report Set instances may not be installable, see Installation Requirements for Each Object Type.

This section contains the following topics:

- Installing the Report Set as a Whole with All Programs Checked In
- Installing the Report Set as a Whole with Some Programs Checked Out
- Installing a Single Program Instance in the Report Set

Installing the Report Set as a Whole with All Programs Checked In

To install a Report Set as a whole, install its Work Area and include the Report Set in the installation.

If all the Programs in the Report Set are checked in, the system installs the whole Report Set and leaves the Report Set itself and all its Programs checked in. You can look at the installation log to see the details.

See Installing a Work Area and Its Objects for further information.

Installing the Report Set as a Whole with Some Programs Checked Out

If you install a Report Set instance and some of its Program instances are checked out by you and noone else, the system checks them in, checks in the Report Set instance and its source definition, and installs the Report Set instance.

If you install a Report Set instance and some of its Program instances are checked out by you and some by someone other than you, and you do not have the LSH Checkin Admin role, the system checks in and installs only the Program instances that were checked out by you. It does not check in the Program instances checked out by others, and it does not check in the Report Set instance or definition or install the Report Set instance.

To see which Programs were and were not installed, look at the installation log file.

> **Note:**
>
> If a user with the LSH Checkin Admin role installs a Report Set, all Program instances in the Report Set are checked in during the installation.

Installing a Single Program Instance in the Report Set

When you install a single Program instance in a Report Set, the system does the following:

- Checks in the Program instance. its mappings, and its source Program definition. (If the Program definition is checked out by a user other than the user installing the Program instance, the installation fails.)
- Checks in the Report Set Entry to which the Program instance is assigned and all Report Set Entries above it in the hierarchy

> **Note:**
>
> The Report Set definition remains checked out.

- Installs the Program instance
- Installs the Report Set Entries, regardless of the setting of the Omit From Install flag
- Installs the Report Set instance
- Installs the Table instances to which the Program's Table Descriptors are mapped, if they are not already installed

No other Program instances in the Report Set are checked in or installed.

You can install a Program instance in a Report Set in several different ways:

- In the Report Set Structure view, select one of the Report Set Entries assigned to the Program instance, select **Install Program** from the drop-down list, and click **Go**.
- In the Report Set Program View, select the Program instance, select **Install** from the drop-down list, and click **Go**.
- In the Program instance's Properties screen, click **Install**.

Validating Report Set Definitions and Outputs

A Report Set may have many Report Set Entries and Program instances in different stages of development at the same time, with reports on data that may become stable at different times. Each of these—Report Set Entries, Program instances, and outputs—has a validation status. You can promote different parts of the Report Set to a new validation status at different times.

Because of this complexity, Oracle LSH provides several tools and other functionality to help you manage Report Set validation:

- **Output reuse**. Each time you run a Program in a Report Set, the system checks if a duplicate output already exists. If it does, the system does not generate the output again, so that any validation work you have done on the output is still valid. See Output Reuse for further information.
- **Validation Status**. Report Set instances and definitions both have a validation status like the validation status of other objects. There are some special rules for Report Sets; see Report Set Validation Status.
- **Summary Output Validation Status**. Report Set instances have an additional status that is automatically calculated for every Report Set Entry and the Report Set instance as a whole; see Summary Output Validation Status.

There are two basic processes you can use to validate a Report Set output:

- Validate the content of each report output manually according to your company standards; see Output-Oriented Validation.
- Validate each Program definition and instance according to your company standards. The Execution Setup inherits its validation status from the Program instance, and the output inherits its validation status from the Execution Setup; see Definition-Oriented Validation.

Oracle LSH supports both approaches with a flag for each Program definition; see Program Output Validation Flag. It is possible to use both approaches within a single Report Set.

See Validating Objects and Outputs for general information about validation.

> **Note:**
>
> You cannot associate validation supporting information and outputs with Report Set Entries. However, you can associate validation supporting information with Program definitions and Program instances inside a Report Set, and for the Report Set as a whole.

This section contains the following topics:

- Output Reuse
- Program Output Validation Flag
- Report Set Validation Status
- Summary Output Validation Status
- Output-Oriented Validation
- Definition-Oriented Validation
- Changing Validation Status

Output Reuse

Each time you run a Program in a Report Set, either directly or through the Report Set, the system checks if a duplicate output already exists. An output is a duplicate if it is generated by the same version of the same Program instance with the same Parameter values on data of the same currency and blinding status.

If the current Program execution would produce an output that is a duplicate of an existing output, the system reuses the existing output in the Report Set unless it has a validation status of Retired.

If all of a Program's Planned Outputs have reusable non-Retired duplicate outputs, the system does not execute the Program at all during the Report Set execution. If one or more Planned Outputs assigned to a Report Set Entry that is included in the execution does not have a reusable duplicate output, the system executes the Program and generates all outputs, but sets the new duplicate outputs' validation status to Retired.

If the system cannot execute a Program instance because it is checked out, or if it fails execution, the Report Set cannot reuse existing outputs because the Program version is different. The Report Set uses the In Progress OTD file as a placeholder for the ungenerated output.

The Jobs screen shows all subjobs that actually ran. The Report Set log file shows which subjobs produced the outputs that were actually used and lists the outputs.

- Report Set Entry Properties and Reuse

Report Set Entry Properties and Reuse

Title- and format-related Report Set Entry properties can be passed to the Program instance assigned to that Report Set Entry. The same Program instance may have multiple Planned Outputs assigned to different Report Set Entries.

Oracle LSH tracks these Report Set Entry-specific values separately from other Parameter values and treats them differently in its duplicate output logic. If one of these values changes, only the output related to the Report Set Entry with the changed value is regenerated. All other outputs—assigned to other Report Set Entries—are reused if a duplicate non-Retired output exists.

For example, if Program A has Planned Outputs assigned to Report Set Entries 1.1, 1.2, and 1.3, and you enter a new value for Title or Font Size in Report Set Entry 1.1 only, and all other Parameter values and the data currency and Program instance version are the same since the last time the Program instance was executed, then the existing outputs for Report Set Entries 1.2 and 1.3 are reused.

> **Note:**
>
> You can force the system to produce and use new outputs for the Report Set Entries included in a submission, even if they are duplicates, by setting the **Force Execution** flag to **Yes** in the Execution Setup.

Program Output Validation Flag

The flag **Force Output Validation Status to 'Development'** is part of every Program definition. If selected, outputs of instances of this Program definition are always created with a validation status of Development. Use this setting if you plan to validate the contents of the output.

If deselected, the outputs inherit the validation status of the Execution Setup that produced them, which in turn can inherit its validation status from the Program

instance. Use this setting if you plan to validate the source data and the Program, and allow the output to inherit its validation status from the Program's.

Your company can determine the default setting using an Oracle profile; see "Setting the Default Value for: Force Output Validation Status to Development" in the chapter on setting profile values in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

You can use both approaches in a single Report Set for different Programs.

Report Set Validation Status

Oracle LSH enforces the following validation rules for Report Sets, their Report Set Entries, and their Program instances:

- You can change the validation status of any Report Set Entry without affecting the validation status of other Report Set Entries either above or below it in the Report Set structure.
- A Report Set or Report Set Entry cannot have a higher validation status than any of the Program instances it contains.
- A Report Set cannot have a higher validation status than any of its Report Set Entries.
- By default, the system calculates the validation status for a Report Set and for a Report Set volume, if any, as the lowest status of all the Report Set Entries it contains.

The system cascades a validation status promotion as follows:

- **Cascade Promotion from Report Set to Report Set Entries**. If you try to promote a Report Set to a higher validation status and all the Program instances in the Report Set are at that status (or higher), but all the Report Set Entries are not, you have the option of cascading the higher status to all the Report Set Entries as part of the Report Set promotion. If you choose not to cascade, the operation fails, because a Report Set cannot have a higher validation status than its Report Set Entries.

  For example, if you want to promote a Report Set from Development to Quality Control (QC), and all of its Program instances are already at QC but some of its Report Set Entries are still at Development, you must promote the Report Set Entries to QC in order to promote the Report Set as a whole to QC. You can do this through a cascade operation as part of the Report Set promotion.
- **Cascade Promotion from Parent to Child Report Set Entries**. If you promote a Report Set Entry that has one or more child (or grandchild, etc.) Report Set Entries with a lower validation status, you receive a message offering you the opportunity to cascade the promotion to all its children. If you choose to proceed, the operation upgrades all child (and grandchild, and so on) Report Set Entries.

  If you choose not to cascade, the operation will proceed for just the parent Report Set Entry.
- **Cascade Promotion from Program Instance to Definition**. As with Program instances outside Report Sets, you can cascade a validation promotion from the Program instance to its definition. If another user has checked out the definition, the operation fails.

- **Cascade Promotion from the Report Set Output to Contributing Outputs in the Same Job**. When you promote a Report Set PDF output, if any of the individual outputs included in it have a validation status lower than the one to which you are promoting the cumulative output, you receive a message asking if you would like to promote the contributing outputs produced in the same job. You must validate all or none of the PDF outputs.

  For example, if you choose to run only one Program instance during the Report Set execution, and that Program instance produces multiple outputs, you can upgrade the validation status of all outputs produced by the Program at the same time.

  > **Note:**
  >
  > Because you are forced to validate all or none of the contributing PDF outputs, if you plan to validate a report set output, be sure to selectively remove from the job any Report Set Entries that should not be validated.

Oracle LSH enforces the following behavior for the Report Set and its components at the following validation statuses:

- **Development**. When you create a new Report Set Entry, either by defining it or by copying it from another Report Set, it automatically receives a validation status of Development. Anyone with Modify privileges on the Report Set can work on it. Users can check out a Program definition through an instance of it in the Report Set if they have Modify privileges on the Program definition.
- **Quality Control**. When a Report Set Entry's validation status is set to Quality Control, a user must have the Modify QC privilege on Report Sets to modify the Report Set Entry in any way, including the addition or removal of a Program instance and Planned Output assignment. A user must have the Modify Validation Status QC privilege on Report Sets to set the validation status back to Development.
- **Production**. When a Report Set Entry's validation status is set to Production, a user must have the Modify Production privilege on Report Sets to modify the Report Set Entry in any way, including the addition or removal of a Program instance and Planned Output assignment. A user must have the Modify Validation Status Production privilege on Report Sets to set the validation status back to Development.
- **Retired**. You cannot set the validation status of a Report Set Entry, or a Program instance in a Report Set, to Retired. If you no longer want to use a Report Set Entry or Program instance, remove it from the Report Set.

Summary Output Validation Status

In addition to the standard validation status (VS), each Report Set Entry also has a Summary Output Validation Status (SOVS). You can use this status to track which Report Set Entries do not yet have a Planned Output assigned and, if you are validating outputs manually, which outputs have not yet been promoted to a new validation status.

The system automatically calculates the SOVS for each Report Set Entry based on the validation status of its actual "current" output (if any) and on the SOVS of its child Report Set Entries (if any).

The Summary Output Validation Status can have the following values:

- **Null**. The system displays no SOVS if a Report Set Entry has a Planned Output assigned but no usable current output, or if the assigned Program instance is checked out.
- **Unassigned**. If a Report Set Entry is not marked as a placeholder and does not have a Planned Output or a Narrative assigned, the SOVS is Unassigned.
- **N/A (Not Applicable)**. If a Report Set Entry has no child Report Set Entries and either is marked as a placeholder or has a Narrative assigned, its SOVS is N/A. If a Report Set Entry does have one or more child Report Set Entries, its SOVS is N/A only if it meets both of the following conditions: all its and, in addition,
  - The SOVS of all its child Report Set Entries is N/A.
  - It is marked as a placeholder or has a Narrative assigned.

  > **Note:**
  >
  > Although this status supports a practice of not including Narratives and Planned Outputs in the same Report Set Entry, it is in fact possible to have both. Be aware that if you assign a Narrative to a Report Set Entry before you assign a Planned Output, the SOVS will be N/A instead of Unassigned.

- **Development**. If any of a Report Set Entry's child Report Set Entries has an SOVS of Null or Unassigned, the parent Report Set Entry's SOVS is Development.

  In addition, a Report Set Entry inherits the SOVS of its current output (if any) or any of its child Report Set Entries (if any), whichever has the lowest SOVS. If any of these is Development, the Report Set Entry is also set to Development.
- **Retired**. If the current output of a Report Set Entry is manually set to Retired, the Report Set Entry's SOVS is also set to Retired. When the Report Set Entry's Program generates a new output, the system changes the Report Set Entry's SOVS to reflect that output's validation status.
- **Quality Control (QC)**. A Report Set Entry has an SOVS of Quality Control if the lowest SOVS of any of its child Report Set Entries or its current output (if any) is Quality Control.
- **Production**. A Report Set Entry has an SOVS of Production if the lowest SOVS of any of its child Report Set Entries or its current output (if any) is Production.

The Report Set definition inherits the SOVS of its child Report Set Entry with the lowest SOVS. The Report Set instance inherits its SOVS from the Report Set definition.

The SOVS is displayed only in the context of a Report Set instance in a Work Area, not for the Report Set definition in its Application Area or Domain.

Output-Oriented Validation

In the output-oriented validation approach, you validate an actual report, including all its data, using the same validation standards your company has always used. The system continues to include the report you have validated in the Report Set as long as you continue to run the same Program instance version with the same currency, blinding status, and Parameter values; see Output Reuse for further information.

If you use the same Program instance to generate a new output of data of a different currency or blinding status, or change any other Parameter values, you must validate the resulting new output.

The output-oriented validation process includes the following steps:

1. When you define Programs for use in a Report Set, set the **Force Output Validation Status to 'Development'** flag to **Yes**; see Creating a Program for further information.

   > **Note:**
   >
   > The **Force Output Validation Status to 'Development'** flag is a property of the Program definition, not the instance.

2. Define and test the Report Set structure and Program instances inside the Report Set over time.
3. When the data a particular Program reports on reaches a point where it should be included in a report for internal use or submission, apply a snapshot label—for example, "Interim"—to the relevant Table instances, either through the job that writes the data to the Table instances or directly in the Work Area (Managing Table Instance Snapshot Labels in a Work Area). This step is optional but recommended.
4. Run the Report Set Entries assigned to the Program that runs on the stable data.
5. Validate each resulting output according to your company's policies, first to Quality Control (QC) and then to Production. The system applies the validation status of the output to the Summary Output Validation Status (SOVS) of the corresponding Report Set Entry; see Summary Output Validation Status.
6. Manually upgrade the standard validation status (VS) of the Program instance and definition and the associated Report Set Entry.
7. Follow the same process for other outputs and their Program instances and definitions and Report Set Entries, until the entire Report Set reaches a higher validation status.

> **Note:**
>
> The Report Set output as a whole inherits the validation status of the Program output with the lowest validation status.
>
> The Report Set definition and instance, and each volume defined in a Report Set, inherit the validation status of the Report Set Entry with the lowest validation status they contain.

When the source data for the original Program reaches another point where it should be included in a report for internal use or submission, apply another snapshot label; for example, "Final," and repeat the process.

Definition-Oriented Validation

In the definition-oriented validation approach, you develop standards for validating the Program instances (and their source Program definitions) that generate each Report Set report output. For example, you can use different sets of test data to test whether the report output contains the correct data. When you have successfully validated the Program definition and instance and set their validation status to Production, the Execution Setups of the Program instances inherit the validation status of Production and the outputs they produce also inherit a validation status of Production.

If you run the same version of the Program instance with different data currency, blinding status, or other Parameter values, the new outputs also have a validation status of Production.

The definition-oriented validation process includes the following steps:

1. When you define Programs for use in a Report Set, set the **Force Output Validation Status to 'Development'** flag to **No**; see Creating a Program for further information.

   > **Note:**
   >
   > The **Force Output Validation Status to 'Development'** flag is a property of the Program definition, not the instance.

2. Define and test the Report Set structure and Program instances inside the Report Set over time.
3. Promote each Program instance and the Report Set Entries to which it is assigned to Quality Control, according to your company's policies.
4. Promote each Program instance and the Report Set Entries to which it is assigned to Production, according to your company's policies.
5. Run the Report Set, or parts of the Report Set, on the appropriate data snapshot(s). The resulting output inherits its validation status from the Execution Setup that generated it, which in turn inherits its validation status from the Report Set instance. If the Report Set instance is set to Production, the output is also set to Production.

> **Note:**
>
> The Report Set output inherits the validation status of the Program output with the lowest validation status.
>
> The Report Set definition and instance, and each volume defined in a Report Set, automatically receive the validation status of the lowest Report Set Entry they contain.

You can manually reset the validation status of an output at any time, subject to your company's policies.

Changing Validation Status

This section contains the following topics:

- Changing the Validation Status of an Output
- Changing the Validation Status of a Report Set Entry
- Changing the Validation Status of a Program Instance in a Report Set

Changing the Validation Status of an Output

When you upgrade the validation status of an output, you must validate the status off all contributing outputs produced during the same job.

To change the Validation Status of a Report Set output, do the following:

1. Navigate to the Report Set output in either the Reports or My Home page:
   - In the Reports tab, navigate the hierarchy that contains the Report Set, then click the icon in the Action column for the output.
   - In the Job Executions section of the My Home page, click the Job ID, then on the Output Name.
2. From the **Actions** drop-down, select **Validation Supporting Info** and click **Go**.
3. Select the **Validation Status** you want from the drop-down list and click **Update**. If any of the contributing Program outputs produced during the same Report Set execution have a validation status lower than the new validation status you are applying, the system displays a message giving the number of such outputs and asks if you want to continue.

   If you choose to continue, all contributing outputs are promoted.

   If you choose not to continue, none of the contributing outputs is promoted.

Changing the Validation Status of a Report Set Entry

To change the Validation Status of a Report Set Entry, do the following:

1. Navigate to the main page for the Report Set Entry. For example, from the Report Set structure screen, navigate in the Report Set hierarchy to the Report Set Entry and click its name.
2. From the **Actions** drop-down, select **Validation Supporting Info** and click **Go**.

3. Select the **Validation Status** you want from the drop-down list and click **Update**.

Changing the Validation Status of a Program Instance in a Report Set

To change the Validation Status of a Program instance in a Report Set, do the following:

1. Navigate to the main page for the Program instance. For example, from the Report Set structure screen, navigate in the Report Set hierarchy to a Report Set Entry to which the Program instance is assigned, then click the Program name in the **Assigned Program Instance** column.
2. From the **Actions** drop-down, select **Validation Supporting Info** and click **Go**.
3. Select the **Validation Status** you want from the drop-down list and click **Update**.

About Report Set Planned Outputs

Individual reports in a Report Set are the Planned Outputs of the Program instances associated with the Report Set.

In addition, the system automatically creates two Planned Outputs for the Report Set as a whole:

- The Report Set log file.
- The Report Set itself; all reports on data. By default, the system creates one Planned Output and gives it the same name (not the title) as the Report Set.

  You can define volume breaks and volume names in Report Set Entries. The system produces each volume as a separate file. Volumes are not displayed in the Planned Outputs subtab. See Setting Report Set Entry Properties for further information.

  You can override Volume Break settings in Report Set Entries by defining a value for the Single Volume Output Name Post-Processing Parameter at the Report Set level.
- The table of contents is generated as a separate file if the TOC Parameter is set to **Yes** and the Report Set contains multiple volumes.

Modifying Report Sets

If you have the necessary privileges, you can modify a Report Set either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area. In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the

original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Report Set Instance Properties
- Modifying Report Set Definition Properties

Modifying Report Set Instance Properties

On the Report Set instance's Properties screen, click **Update** to enter changes. Oracle LSH creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must reinstall the Report Set for the changes to take effect.

You can modify the following:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field applies to the instance only. It specifies the Report Set definition to which this Report Set instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Table Descriptors, Source Code, Parameters, and Planned Outputs.
  - Any new Table Descriptors are not mapped.
  - The Report Set's status changes to **Non Installable**.

  If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.

Modifying Report Set Definition Properties

You can go to a Report Set definition's Properties screen in one of the following ways:

- **From the Report Set's Properties screen**: Click the hyperlink of the Report Set definition that appears in the Definition field.
- **From the Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Report Set definition screen, click **Update** to enter changes. Oracle LSH creates a new version of the definition. You can change the following properties:

- Name: See Naming Objects for further information.

- Description: See Creating and Using Object Descriptions for further information.

  You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

- Modifying the Report Set Structure
- Modifying Report Set Entries
- Modifying Programs

Modifying the Report Set Structure

You can modify the Report Set's structure in two places:

- **Report Set Structure view**. You can add, remove, copy, and move Report Set Entries here; see Using the Report Set Structure View.
- **Report Set or Report Set Entry Properties screen**. You can also add, remove, copy, and move Report Set Entries in Entries subtab of the Properties screen of the parent Report Set or Report Set Entry; see Defining Report Set Entries.

Modifying Report Set Entries

To modify many aspects of a Report Set Entry definition, you must go to its Properties screen. You can go to the Properties screen by clicking its hyperlink in the Report Set Structure view or in its parent Report Set or Report Set Entry's Properties screen, then do the following:

- Click **Update** to modify a Report Set Entry's properties, change its Program or Planned Output assignment, or its associated narrative(s) see Defining Report Set Entries.
- Click the **Entries** subtab to add, remove, or reorder a Report Set Entry's child Report Set Entries.
- Click the **Parameters** subtab to add, remove, or modify Parameters defined in the Report Set Entry for the purpose of value sharing; see Creating Parameters for Sharing Values within the Report Set.
- Click the **Post-Processing** subtab to change the value or properties of the predefined post-processing Parameters; see Setting Post-Processing Parameter Values.
- Click the **Overlay Templates** subtab to change the value or properties of the predefined overlay template Parameters; see Setting Overlay Template Parameter Values.

In addition, you can use the Report Set Structure view to change a Report Set Entry's Program or Planned Output assignment or reorder its child Report Set Entries; see Using the Report Set Structure View.

Modifying Programs

The Program instances contained in a Report Set belong to the Report Set definition. Their mappings belong to the Report Set instance. You modify Programs in a Report Set the same way you modify Programs outside a Report Set; see Defining Programs. You can reach the Program instance screen in a Report Set by clicking its hyperlink in any of several places:

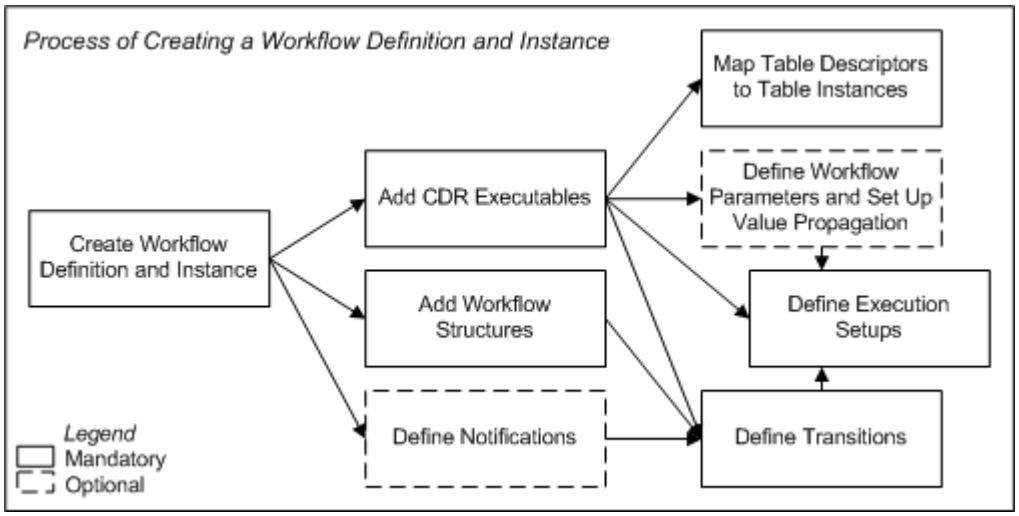
- Click its hyperlink in the Report Set Structure view next to one of the Report Set Entries to which it is assigned.
- Click its hyperlink in the Report Set Program view.
- Click it hyperlink in the Properties screen of one of the Report Set Entries to which it is assigned.

You can map the Table Descriptors of a Program in a Report Set in several different ways; see About Programs in Report Sets.

10

Defining Workflows

Figure 10-1 Process of Creating a Workflow Definition and Instance



This section contains information on the following topics:

- About Workflows
- Creating a Workflow
- Using the Workflow Properties Screen
- Adding Executables
- Adding Workflow Structures
- Defining Notifications
- Defining Transitions
- Defining Workflow Parameters
- Workflow Planned Outputs
- Installing Workflow Instances
- Modifying Workflows

About Workflows

Workflows allow you to create a single automated process that includes multiple steps, or activities, with conditional branching. The activities can include Oracle Life Sciences Data Hub (Oracle LSH) executables—Load Sets, Programs, Report Sets, and Data Marts—and Structures such as Fork, And, and Or.

Using a single Workflow you can, for example:

- Load data from SAS and Oracle Clinical into Oracle LSH, write Programs to combine, transform and report the data, combine the results into a Report Set, and notify the appropriate people that the Report Set is available
- Use the results of one Program in a subsequently executed Program in the same Workflow
- Use Programs that run on different applications, such as SAS and PL/SQL, in the same Workflow
- Generate a Report Set or Data Mart intended for submission to a regulatory agency and send Notifications to the appropriate people requesting their formal approval before sending the Report Set or Data Mart to the regulatory agency
- Include Program outputs in the zipped file of a Data Mart

- Workflow Components
- Workflow Rules
- Execution
- Reports on Workflow Definitions and Instances

Workflow Components

A Workflow includes the following components, which you must define. Executable instances in a Workflow, Workflow Structures, and Notifications are all called Workflow **activities**.

- **Oracle LSH Executable Object Instances**. To execute a Load Set, Program, Report Set, or Data Mart as part of a Workflow, you must create an instance of it in the Workflow. See Adding Executables.
- **Workflow Structures**. You can add predefined structural activities such as Fork, And and Or to control Workflow execution. See Adding Workflow Structures.
- **Notifications**. Notifications are messages sent to Oracle LSH users either simply to convey information or to request approval. You can link a Notification to the completion of an activity in the Workflow. For example, when a Load Set execution completes you can request approval from one or more people. The Workflow waits while these people verify that the Load Set successfully loaded the correct data. Only after receiving approval from one or all recipients (depending on how you define the Notification) does the Workflow proceed to the next activity. You can also use a Notification to alert a group of people that a report has been successfully generated on fresh data. See Defining Notifications.
- **Transitions**. Transitions define the condition, if any, required to move from one activity to the next. You must define a Transition between each sequential pair of activities. See Defining Transitions.
- **Workflow Parameters**. You can define Parameters directly in the Workflow for the purpose of passing their value (which is set in the Execution Setup definition or at runtime) to input Parameters of the executables within the Workflow. You can also set up value propagation from the output Parameter of one Program to the input Parameter of another Program executed later in the Workflow. See Setting Up Parameter Value Propagation.

Workflow Rules

Workflows must conform to a number of rules to be valid and installable.

Execution

During Workflow execution the system detects when one activity has ended and triggers the beginning of the next if the necessary condition has been met, according to your definition of the Workflow activities and Transitions.

The system prevents access to data produced by Programs within a Workflow to Programs outside the Workflow until execution of the entire Workflow has been completed (see Refresh Groups).

The system uses Oracle Workflow to create and execute Workflows.

You define Execution Setups for Workflows the same way you do for other executables. See Creating, Modifying, and Submitting Execution Setups for further information.

Reports on Workflow Definitions and Instances

From the **Actions** drop-down list, you can generate reports that provide information on a Workflow definition or instance; see System Reports for information.

Creating a Workflow

When you create a Workflow in a Work Area, you are actually creating an instance of a Workflow definition.

To create a new Workflow instance:

1. In a Work Area, select **Workflow** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Workflow screen.
3. Choose one of the following options:
   - **Create a new Workflow definition and instance**. Choose this option if no Workflow definition exists that can meet your needs, either as it is or with some modification.
   - **Create an instance from an existing Workflow definition**. Choose this option if a Workflow definition already exists that meets your needs.

     If you can adapt an existing Workflow definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.
4. Depending on your choice, follow one of the following sets of instructions:

- Creating a New Workflow Definition and Instance
- Creating an Instance of an Existing Workflow

Creating a New Workflow Definition and Instance

When you select **Create a new Workflow definition and instance** in the Create Workflow screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.
   - **Description**. See Creating and Using Object Descriptions.
2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Workflow.

   The system opens the Properties screen for the new Workflow instance.
4. Define the Workflow details. It may help you to draw a diagram on paper or white board of the Workflow so you can see which executables, Structures, Notifications and Transitions you need to define. Begin by adding executables and Workflow Structures. See:
   - Adding Executables
   - Adding Workflow Structures
   - Defining Notifications
5. Next, add Transitions and Workflow Parameters and create at least one Execution Setup. See:
   - Defining Transitions
   - Defining Parameters
6. After you have finished setting up Parameter value propagation, define at least one Execution Setup. See Creating, Modifying, and Submitting Execution Setups.
7. Click **Check In**. The system checks in Version 1 of the Workflow definition.
8. Install the Workflow instance (see Using, Installing, and Cloning Work Areas).
9. Validate both the definition and the instance according to your company's policies.

Creating an Instance of an Existing Workflow

If you use an existing Workflow as a definition source, all of its elements, Transitions, Parameters and other characteristics are already defined. See Creating an Instance of an Existing Definition for instructions.

You must go into each executable instance contained in the Workflow and map its Table Descriptors to Table instances. You must map target Table Descriptors to Table instances located in the same Work Area as the Workflow instance. See Mapping Table Descriptors to Table Instances for instructions.

> **Note:**
>
> If you have developed the executables outside the Workflow there may be mapping conflicts because only one Program instance can write to a Table instance. See Mapping Table Descriptors within a Workflow for further information.

Using the Workflow Properties Screen

If you are working in a Work Area, you see the properties of both the Workflow instance and the Workflow definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See Modifying Workflows for information on modifying Workflows.

See also Figure 10-1

This section contains the following topics:

- Instance Properties
- Definition Properties
- Buttons

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Definition:** This field specifies the Workflow definition to which this Workflow instance points.

  You can upgrade to a new version of the same definition. See Upgrading to a Different Definition Version from an Instance.
- **Version:** This field displays the current version number of the Workflow instance.
- **Version Label:** This field displays the version label, if any, for the current Workflow instance version.

  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.
- **Validation Status:** This field displays the current validation status of the Workflow instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Workflow: Installable or Non Installable. See Installation Requirements for Each Object Type.

Definition Properties

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to add, remove, or modify activities, structures, or transitions. However, you can map Table Descriptors without checking out the definition. See Understanding Object Versions and Checkin/Checkout for further information.
- **Latest Version:** If set to **Yes**, this Workflow instance is pointing to the latest version of the Workflow definition. If set to **No**, this Workflow instance is pointing to an older version of the Workflow definition.
- **Checked Out By:** This field displays the username of the person who has the Workflow definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Validation Status:** This field displays the current validation status of the Workflow definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Workflow in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Workflow: Installable or Non Installable. See Installation Requirements for Each Object Type.

Buttons

From a Workflow instance in a Work Area, you can use the following buttons:

- **Install:** Click **Install** to install the Workflow instance, including mapping target Table Descriptors and installing mapped target Table instances; see Installing Workflow Instances. For a list of reasons a Workflow instance may not be installable, see Installation Requirements for Each Object Type.
- **Submit:** Click **Submit** to run the Workflow instance. Before you can run the Workflow, you must install it and create an Execution Setup for it (select **Execution Setups** from the **Actions** drop-down list).
- **Update:** Click **Update** to modify the Workflow instance properties. See Modifying Workflow Instance Properties.
- **Check In/Out and Uncheck:** Click these buttons to check out, check in, or uncheck the Workflow definition. Different buttons are displayed in the Workflow Definition Properties section depending on the Checked Out Status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of the person who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.

Adding Executables

This section includes the following topics:

- About Executables as Workflow Activities
- Adding an Executable to a Workflow
- Mapping Table Descriptors within a Workflow

About Executables as Workflow Activities

Oracle LSH executables—Load Sets, Programs, Report Sets, and Data Marts—are the substance of a Workflow; the purpose of a Workflow is to execute them as a single process, with contingencies for the success or failure of the execution of each one.

You must add executables to the Workflow before you can define Transitions between them or Parameter value propagation. However, you can add executables to the Workflow over time and add or modify Transitions and Parameter value propagation as necessary.

Although you have the option of creating a new executable from within the Workflow, Oracle recommends that you either use an existing validated definition of a Load Set, Program, Report Set, or Data Mart, or create a new one in a Work Area first so that you can test it independently before creating an instance of it in the Workflow. However, this may lead to mapping conflicts; see Mapping Table Descriptors within a Workflow for further information.

> **Note:**
>
> Oracle does not support multiple concurrent users developing a Report Set inside a Workflow. If more than one person will work on the Report Set at a time, develop the Report Set directly in a Work Area and add it to the Workflow when it is ready.

Adding an Executable to a Workflow

To add an Oracle LSH executable to the Workflow, go to the Table of Contents tab and do the following:

1. From the Add drop-down list, select the type of executable you want to add.
2. Click Go. The system opens the Create screen for the object type you selected.
3. Choose one of the following.
   - **Create an instance of an existing object definition**. See Creating an Instance of an Existing Definition for instructions.

     > **Note:**
     >
     > Oracle recommends this option because you can run and test the executable independently before including it in the Workflow.

   - **Create a new object definition and instance**.
4. Click **Apply**.

See Finding an Appropriate Definition and Creating and Reusing Objects for general information on reusing object definitions.

Mapping Table Descriptors within a Workflow

When you add a Load Set, Program, Report Set or Data Mart instance to the Workflow through a Workflow instance, you cannot map Table instances to Table Descriptors from the executable's screen. You must use the **Table Instances from Existing Table Descriptors** job from the Actions drop-down list on the Workflow's Properties screen.

See Adding a New Target Table Descriptor for further information.

If you create a Load Set or a Program in the same Work Area where your Workflow instance is located, and then create a new instance of that Load Set or Program inside the Work Area, the system does not allow you to map the instance in the Workflow to the same target Table instances. Only one executable can write to any particular Table instance. You can do either of the following:

- **Create new Table instances** and map them to the Program or Load Set instance that is contained in the Workflow. You can use the Table Instances from Existing Table Descriptors job in the Actions drop-down list to create the Table instances and map them automatically.
- **Unmap the target Table instances** from the instance of the Program or Load Set that is contained directly in the Work Area.

  If you unmap before adding the Program or Load Set to the Workflow, and if the Table instances have the same name as the Table Descriptors, you can run the Automatic Mapping By Name job in the Actions drop-down list after you add the instances to the Workflow.

See Mapping Table Descriptors to Table Instances for further information.

Adding Workflow Structures

This section contains the following topics:

- About Workflow Structures
- Types of Workflow Structures
- Adding Structures

About Workflow Structures

Workflow Structures are predefined activities that you use to control the execution of the Workflow. They can detect the completion status of the previous activity through their input Transition(s) and fire the next one or more activities as specified for their type.

You must add the number of Structures you need; if you need two Forks, for example, you must add two Forks. You need to be able to distinguish the two Forks from one other when you define the Workflow's Transitions. Therefore, give them unique names; for example, append the name of the preceding activity, such as Program_A_Fork.

Structures can be followed only by unconditional Transitions.

When you create a Workflow, the system automatically adds one Start Structure and one of each End Structure to the Workflow. You must add Transitions to connect the Start Structure to the first Workflow activity and to connect one End-type Structure to the last activity on each branch.

Types of Workflow Structures

Each Workflow Structure has its own function, as follows:

- **And:** An And activity has multiple input activities and a single output activity. It synchronizes events; it waits for all incoming Transitions to be completed before proceeding with the next Transition.
- **Fork:** A Fork activity has a single input activity and multiple output activities, creating two or more branches. The subsequent activities run in parallel.
- **Or:** An Or activity has multiple inputs and a single output. It fires the next Transition as soon as any one of its input activities completes.
- **Start:** All Workflows must contain a single Start Structure as their first activity. A Workflow can contain only one Start. If you need to begin with more than one activity—for example, two or three SAS Load Sets, each loading a different data set—use a Fork immediately after the Start, with an unconditional Transition to each Load Set after the Fork.
- **End_Success:** End_Success ends Workflow execution with a status of Success. Use this Structure at the end of the successful completion of the Workflow. If you have multiple branches with parallel activities, use an And Structure before End_Success so that they must all complete successfully before Workflow execution ends with success.
- **End_Failure:** End_Failure ends Workflow execution with a status of Failure. Use this Structure following a Transition with a condition of Failure, after each executable. If the executable fails to execute properly, execution of the Workflow ends with an error.

  Use this Structure also following a Notification of type Approval, after a conditional Transition whose condition is Failure. If an Approval times out—its recipients do not either approve or reject it within the defined timeout period—the system interprets the timeout as a failure.
- **End_Warning:** End_Warning ends Workflow execution with a status of Warning.

  You can use this Structure after a conditional Transition whose condition is Warning.

Adding Structures

You can add Workflow Structures in the following places in the Workflow user interface:

1. In the **Activities** subtab, choose a Structure from the **Add** drop-down list and click **Go**. The Create Workflow Structure screen opens.
2. From the Structure Type drop-down list, select the type of Structure you want to add.
3. Enter a name for the Structure. The name will help you distinguish this structure from others of the same type, so that you create the transitions you intend.

4. Click Apply.

> **Note:**
>
> Only one Start structure is permitted.

Defining Notifications

This section includes the following topics:

- About Notifications
- Using Approvals
- Creating a Notification
- Modifying Notifications

About Notifications

A Notification is a message that you can define as an activity in a Workflow to be sent to recipients you specify after the completion of another Workflow activity. A Notification can include the following:

- a subject line
- a text message
- a link to an Oracle LSH output
- a request for approval or rejection

A Notification appears on the Oracle LSH My Home screen of each recipient. Each Oracle LSH user can also set a user preference to receive Notifications as email. See "Using Notifications" in the *Oracle Life Sciences Data Hub User's Guide* for instructions.

You can create a set of standard Notification definitions for reuse.

- Types of Notifications

Types of Notifications

There are two types of Notifications:

- **FYI (For Your Information)**. FYI Notifications pass information to the recipients. They can contain text and a hyperlink to an Oracle LSH output. You must specify one or more recipients by user group and role.
- **Approval**. Approvals pass information to the recipients and also request action from the recipients. The system waits for a response before proceeding to the next activity in that branch of the Workflow. You can define the Notification to require action from one or from all recipients.

Using Approvals

You can use an Approval as a manual checkpoint in the Workflow's business process—for example, to ensure that the system loaded the correct lab data before generating a report on the data—or as a record that a report output has been reviewed and approved (see Writing Text for Approvals).

In the Notification you specify whether approval is required by one or all of the recipients to proceed to the next Workflow activity. You also specify a timeout period. You can specify a backup group of recipients to receive the Notification if the timeout period expires, plus a timeout period for the backup recipients.

When a user receives an approval-type Notification, it includes two buttons labeled Approve and Reject, respectively, but the approval or rejection is not of the Notification itself. You must make it clear to the recipient in the text what will happen if he or she clicks each button.

You must define two Transitions following a Notification, one for each condition: Success and Failure. The system interprets an approval by the required number of recipients as success, and a rejection or a timeout as a failure.

Success and Failure Conditions: In the Workflow, you must define two Transitions following each Approval Notification, one each for success and failure. The system interprets an approval as a success, and a rejection or a timeout as a failure.

Creating a Notification

When you create a Notification in a Work Area, you are actually creating an instance of a Notification definition.

To create a new Notification instance:

1. In a Work Area, select **Notification** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Notification screen.
3. Choose one of the following options:
   - **Create a new Notification definition and instance**. Choose this option if no Notification definition exists that can meet your needs, either as it is or with some modification.
   - **Create an instance from an existing Notification definition**. Choose this option if a Notification definition already exists that meets your needs.

     If you can adapt an existing Notification definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.
4. Depending on your choice, follow one of the following sets of instructions:
   - Creating a New Notification Definition and Instance
   - Creating an Instance of an Existing Definition

- Creating a New Notification Definition and Instance

- Creating an Instance of an Existing Notification
- Specifying Notification Recipients
- Defining a Link to a Planned Output
- Writing Notification Messages
- Defining Notification Parameters

Creating a New Notification Definition and Instance

When you select **Create a new Notification definition and instance** in the Create Notification screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.

     > **Note:**
     >
     > The Name does not appear on the actual notification sent to recipients. Recipients see the Subject instead. Definers searching for a Notification definition to use can see the Name.

   - **Description**. See Creating and Using Object Descriptions.
   - **Subject**. Enter the summary text recipients see, up to 80 characters including text variable values. See Creating and Using Text Variables.
   - **Type**. Select FYI (For Your Information) or Approval. See Types of Notifications.
   - **Priority**. In Oracle LSH Release 2.5 this field has no effect.
   - **Primary Recipient Timeout**. This field applies only to Notifications of type Approval. Specify the days, hours, and/or minutes from the time the system sends the Notification to the time the Notification times out for the primary recipients.
   - **Backup Recipient Timeout**. This field applies only to Notifications of type Approval. Specify the days, hours, and/or minutes from the time the Notification times out for the primary recipients to the time the Notification times out for the backup recipients. If you do not plan to add backup recipients, enter a zero (0) in each field.
2. If you selected Approval as the type, select a value for **Approval By**:
   - **All Recipients**. The system moves to the next Workflow activity after the success Transition only after all recipients have approved the Notification.
   - **Any Recipients**. The system moves to the next Workflow activity after the success Transition as soon as any one recipient approves the Notification.

     In the case of All Recipients, if a single recipient rejects the Notification, and in the case of Any Recipients, if all the recipients reject the Notification, the system moves to the next Workflow activity after the error Transition.

     Timeout and Rejection are both failures and the system moves to the workflow after the error transition in both the cases.

3. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
4. Click **Apply** to save your work and continue defining the Notification.

   The system opens the Properties screen for the new Notification instance. You must click **Update** to add recipients or links to outputs.
5. Define the Notification details. See:
   - Specifying Notification Recipients
   - Defining a Link to a Planned Output
   - Writing Notification Messages
   - Defining Notification Parameters
6. Click **Check In**.

   The system checks in Version 1 the Notification definition. The definition is located directly in the current Application Area. The instance is located in the Workflow.

Creating an Instance of an Existing Notification

If you create an instance of an existing Notification, its Recipients, Parameters, Message, and other attributes are already defined. You can modify them as necessary in the instance. See Creating an Instance of an Existing Definition for instructions.

Specifying Notification Recipients

You specify recipients by their user group and role. The system sends the Notification to all users who have the role you select in the user group you select. You can specify any number of combinations of user groups and roles.

To add recipients:

1. Click **Update**. The system refreshes the screens with enterable fields.
2. From the **Group** drop-down list, select the user group to whom you want to send the Notification.
3. From the **Role** drop-down list, select the role required for Notification recipients.
4. From the **Recipient Type** drop-down list, select either Primary or Backup.
   - **Primary** recipients receive the Notification when it is first generated by the Workflow.
   - **Backup** recipients receive the Notification only if the first group of recipients do not approve or reject the Notification before the timeout period ends.
5. Click **Add Recipient**. The system adds the user group and role combination to the recipient list.
6. Click **Apply**.

If you want to specify one or more recipients by name, you must create a user group specifically for this purpose and add the users to the group. Creating user groups requires administrative privileges.

Defining a Link to a Planned Output

In the Notification definition you must define a link to a Planned Output of an executable contained in the Workflow. At runtime, the system generates a link to the actual output generated from the Planned Output, and puts the link into the Notification in a reserved area of the Notification.

To define a link to a Planned Output:

1. Click **Update**. The system refreshes the screens with enterable fields.
2. Click **Add Planned Output**. The **Search and Add Planned Outputs** screen appears.
3. From the **Activities in the Workflow** drop-down list, select the executable to whose output you want to create a link.
4. In the **Output Name** field, enter the name or title of the Planned Output to which you want to create a link. You can use special characters; see "Using Special Characters" in the *Oracle Life Sciences Data Hub User's Guide* for instructions.
5. Click **Go**. The system performs the search and displays the results.
6. Select the Planned Output you want by clicking its **Select** checkbox and clicking the **Select** button.

Writing Notification Messages

This section includes the following topics:

- Creating and Using Text Variables
- Writing Text for Approvals

Creating and Using Text Variables

The Message is the text content of the Notification. The message and Parameters are part of the Notification definition, and you must click the hyperlink to the definition to define them.

To create the Notification message:

1. Click the hyperlink to the definition located under the Subject line. The Notification definition's Properties screen appears.
2. Click **Update**. The system refreshes the screen and makes the fields enterable.
3. In the Messages subtab, enter text in the **Body** field. You can use the **Insert Parameter** button to insert runtime Parameter values into the body of the text; see Defining Notification Parameters.
4. Click **Apply**.

Writing Text for Approvals

When you write text for approval Notifications, be careful to make it clear to the recipient what a response of Approve or Reject means. For example, use wording such as:

- If you have reviewed this report and believe its contents are accurate, click Approve. If you have reviewed this report and believe its contents are not accurate, click Reject.
- Please review the report on the lab data load at the link below. If the system has loaded the correct data, click Approve. If not, click Reject.

The system does not link the approval or rejection to the actual report. In case of an approval, it proceeds to the step following the success Transition in the Workflow. If rejected, the system proceeds to the step following the error Transition. Therefore, if clicking Approve indicates approval of the report's contents, ensure that the Notification text conveys that understanding.

The system keeps a record of all approvals and rejections.

Defining Notification Parameters

You can save time and ensure consistency and quality by reusing Notification definitions with clear, carefully worded text that includes Parameters as text variables so that you can use them in multiple situations.

The system places the Parameter value in the message at the point where you insert the Parameter, displayed in the Notification definition preceded by an ampersand (&). At runtime the system replaces the backslashes and the Parameter name with the current value of the Parameter.

The Parameter appears in the Execution Setup of the Workflow under the Notification. You can define a Parameter directly in the Workflow that will appear at the top of the Execution Setup and set up value propagation from that Parameter to the Notification Parameter; see Defining Parameters .

For example, you can use the same Notification definition for multiple reports and multiple studies by inserting the following Parameters into the message text:

- Study Name
- Report Title

The person submitting the Workflow for execution enters the correct value for that execution and the system propagates the value to the Notification, so that Notification recipients see the appropriate study and report title.

> **Note:**
>
> If the Parameter value is so long that the subject or body exceeds the maximum size, the system truncates the subject or text. The maximum size of the subject is 80 characters. The maximum size of the body is 32K characters.

Modifying Notifications

The system copies all the attributes of a Notification definition to the instance. Your changes affect only the instance of the Notification in the Workflow, not the source definition. You can modify the following attributes:

- Priority

- Recipients
- Approval Required By All Recipients
- Timeout
- Backup Recipients
- Backup Timeout
- Link to Planned Output

Defining Transitions

This section includes the following topics:

- About Transitions
- Creating Transitions

About Transitions

Transitions specify the sequence of activities in the Workflow and the condition, if any, necessary to proceed from one activity to the next. You must explicitly define the Transition between each sequential pair of activities.

You must add activities (Oracle LSH executables—Load Sets, Programs, Report Sets, Data Marts, and Notifications—and Workflow Structures—Start, And, Or, Fork, and the End Structures—before you can define the Transitions between them.

Each activity (except Start and End Structures) must have a Transition defined immediately before and after it in the Workflow. In addition, each Transition must have an activity defined immediately before and after it.

You can define more than one Transition following a single activity, but each Transition must each have a different condition. To create parallel branches where each branch is the result of the same condition of the same activity, use a Fork activity.

Transitions can be conditional or unconditional:

- **Unconditional Transitions** occur as soon as the first activity has completed. If the first activity is an executable, either a success or failure end status triggers the next activity. You can define an unconditional Transition after either an executable or a structural activity.

  Conditional Transitions can occur only after an executable or a Notification. The executable must return a completion status of success, warning or failure and you must define a conditional transition to handle each completion status.

See Workflow Rules for further information.

Creating Transitions

Create Transitions in the Workflow's Properties screen, Transitions subtab.

1. In the **Transitions** subtab, click **Add Transition**. The system opens the Create Workflow Transition screen.
2. Display the **From** drop-down list. The system displays all the activities you have created for the Workflow.

3. Select the activity that occurs earlier in the Workflow of the two whose Transition you are defining. For example, if you are defining the Transition between the Workflow Structure Start and the first executable, choose Start in the **From** column.
4. From the **To** drop-down, choose the second activity of the two whose Transition you are defining. This activity will occur immediately following the first if the condition is met.
5. From the **Condition** drop-down list, choose the condition you want to apply to the Transition. The choices are:
   - **Error**. If the first activity of the pair ends in failure (or a Notification times out, or is rejected), the system uses this Transition to determine the next activity.
   - **None**. The Transition is unconditional. The second activity is fired regardless of the completion status of the first.
   - **Success**. If the first activity of the pair ends in success (or a Notification of type Approval is approved), the system uses this Transition to determine the next activity.
   - **Warning**. If the first activity of the pair ends in warning, the system uses this Transition to determine the next activity.

   > **Note:**
   >
   > If you define an unconditional Transition between two activities, you cannot also define conditional Transitions between the same two activities.

6. Click **Apply**. The system returns you to the Transitions subtab and displays the Transition you just defined in the last row.

Continue adding Transitions until you have covered all those necessary for the Workflow. Each activity (except Start and End ones) should occur at least once in the **From** column and once in the **To** column. If you use Forks or create branches for different outcomes (success or failure), some activities must appear multiple times in either or both columns.

Defining Workflow Parameters

You can define Parameters directly in the Workflow for the purpose of passing their value to the input Parameters of Programs and Report Sets contained in the Workflow. See Defining Parameters and Setting Up Parameter Value Propagation for information.

Workflow Planned Outputs

A Workflow may generate multiple report outputs, but the system uses the Planned Outputs defined for Programs contained in the Workflow to generate them.

The Workflow itself has no Planned Outputs. Outputs are produced by the executable objects under the Workflow. Therefore Planned Outputs are defined directly under the objects.

Installing Workflow Instances

You can install a Workflow instance directly from its Properties screen, using the Install button, or in its Work Area (see Installing a Work Area and Its Objects).

When you install a Workflow instance using the **Install** button on its Properties screen:

- The system checks in the Workflow instance and definition, and also the Table instances in the current Work Area to which the instance is mapped.
- The system checks if the Workflow is installable. If not, the system performs Automatic Mapping by Name on any unmapped target Table Descriptors. If the Workflow is still not installable and there are still unmapped target Table Descriptors, the system creates Table instances in the current Work Area from the target Table Descriptors and maps them.
- The system attempts to install the Workflow instance and its source and target Table instances in the current Work Area. The system displays a success or error message. If the installation fails, the error message displays the name of any objects that were not installable.

> **Note:**
>
> If any of the Table instances or the Workflow definition is not installable, the system cannot install the Workflow instance. See Installation Requirements for Each Object Type for the reasons these objects may not be installable.

> **Note:**
>
> To continue working on the Workflow, check it out.

- Accessing the Log File

Accessing the Log File

To see the log file for the installation, you must go to the Work Area Installation screen, as follows:

1. Click the **Applications** tab. The main Application Development screen opens.
2. Click the name of the Work Area you are working in. The Work Area screen opens.
3. From the **Actions** drop-down list, select **Installation History**.
4. Click **Go**. The system displays the Installation History screen with the log files in chronological order.
5. Click the **View Log** link for the most recent installation attempt or for the date and time that you ran the install process. The system displays the log file.

For information on installation and on reading the log file, see Installing a Work Area and Its Objects.

Modifying Workflows

If you have the necessary privileges, you can modify a Workflow either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area. In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Workflow Instance Properties
- Modifying Workflow Definition Properties

Modifying Workflow Instance Properties

On the Workflow instance's Properties screen, click **Update** to enter changes. Oracle LSH creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must reinstall the Workflow for the changes to take effect.

You can modify the following:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field applies to the instance only. It specifies the Workflow definition to which this Workflow instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Activities, Transitions, Notifications, and Parameters.

- Any new Table Descriptors are not mapped.
- The Workflow's status changes to **Non Installable**.

If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.

Modifying Workflow Definition Properties

You can go to a Workflow definition's Properties screen in one of the following ways:

- **From the Workflow's Properties screen**: Click the hyperlink of the Workflow definition that appears in the Definition field.
- **From the Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Workflow definition screen, click **Update** to enter changes. Oracle LSH creates a new version of the definition. You can change the following properties:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.

  You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.
- Modifying Activities and Transitions
- Modifying Parameters
- Modifying Table Descriptor Mappings

Modifying Activities and Transitions

Activities (executable instances, Notification instances, and Workflow Structures) and Transitions belong to the Workflow definition. You cannot modify them, but you can add and remove them.

Modifying Parameters

Parameters belong to the Workflow definition. You must check out the Workflow definition to modify Parameters.

You can change the following:

- You can add and remove Parameters owned directly by the Workflow, modify their default value and other settings, and change the Parameters to which they propagate their value at runtime.
- You can modify Parameter value propagation relations defined in the Workflow from the output Parameter of one Program to the input Parameter of another Program executed subsequently in the Workflow.

You cannot modify, add, or remove the Parameters that belong to the Programs and other executables inside a Workflow, except to redefine their value propagation relationships. See Setting Up Parameter Value Propagation.

You can change some Parameter values and settings in one or more Execution Setups. Select **Execution Setups** from the **Actions** drop-down list in the Workflow instance in the Work Area. See Creating, Modifying, and Submitting Execution Setups.

Modifying Table Descriptor Mappings

All the Table Descriptors in a Workflow belong to the executable instances it contains, including Load Sets, Programs, Data Marts, and the Program instances contained in Report Set instances in the Workflow.

However, you can use the Actions drop-down item **Table Instances from Existing Table Descriptors** directly from the Workflow, to create Table instances from existing Table Descriptors owned by any Program instance in the Workflow and map them at the same time; see Creating Table Instances from Table Descriptors and Simultaneously Mapping Them.

You can also use Automatic Mapping by Name directly from the Workflow; see Automatic Mapping by Name.

11

Defining Business Areas for Visualizations

This section contains information on the following topics:

- About Visualizations
- Creating a Business Area
- Using the Business Area Properties Screen
- Defining Table Descriptors
- Defining Joins
- Defining Business Area Hierarchies
- Understanding Business Area Source Code
- Setting Business Area Attributes and Parameters
- Defining Oracle Business Intelligence Business Areas
- Defining Generic Visualization Business Areas
- Launching Visualizations
- Installing Business Area Instances
- Modifying Business Areas

About Visualizations

The Oracle Life Sciences Data Hub (Oracle LSH) is integrated with visualization tools, to allow nontechnical users to quickly create onscreen graphical and tabular displays and interactive dashboards of Oracle LSH data. Any Oracle LSH data can be made available for this purpose—freshly loaded source data or merged and transformed data.

You make data available to the visualization tool by defining one or more Business Areas in Oracle LSH. You define Table Descriptors in a Business Area and map them to Table instances. The visualization tool can read data contained in the Table instances to which the Table Descriptors are mapped. An Oracle LSH Business Area corresponds to an Oracle Business Intelligence Enterprise Edition (OBIEE) Repository and a particular visualization can see data through a single Business Area. Your company may support other visualization tools and an Oracle LSH Business Area of that type may correspond to an element in those visualization tools.

You can also define Joins between the Table Descriptors and Hierarchies (drill-down structures) of Columns in the same Table Descriptor or in joined Table Descriptors. Joins and Hierarchies help determine how visualizations can compare and organize data; see Defining Joins and Defining Business Area Hierarchies.

When you launch a Visualization tool from a Business Area or from the Visualizations subtab of the Reports tab, you can choose to see non-current and blinded data, if you have the appropriate privileges, through the Launch Settings screen. If you open

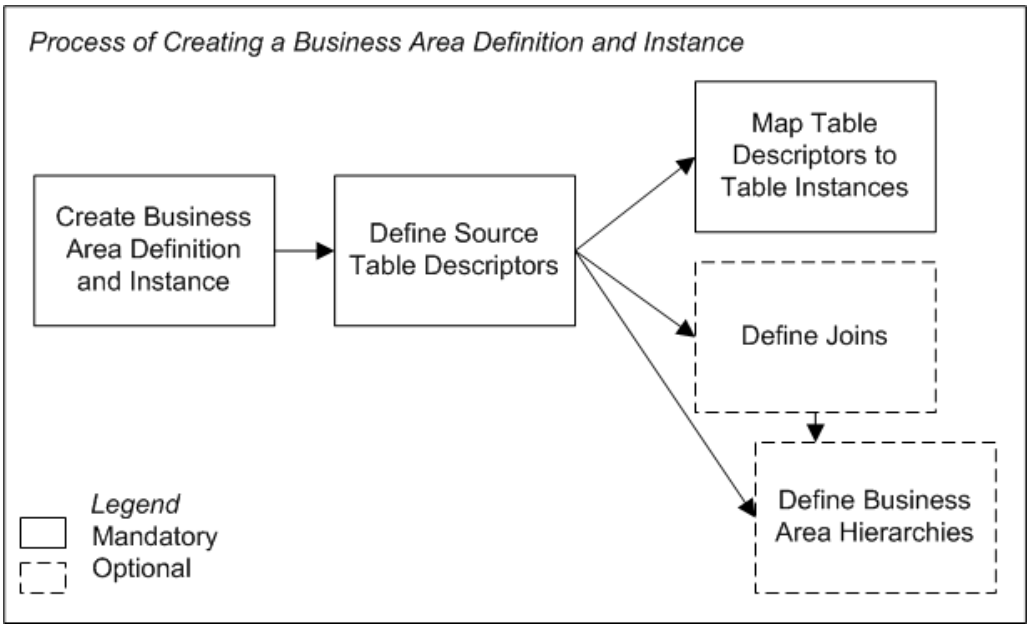
OBIEE directly (not from within Oracle LSH) you can view only current and not blinded data regardless of your privileges.

Reports on Business Area Definitions and Instances: From the **Actions** drop-down list, you can generate reports that provide information on a Business Area definition or instance; see System Reports for information.

Creating a Business Area

When you create a Business Area in a Work Area, you are actually creating an instance of a Business Area definition.

Figure 11-1 Process of Creating a Business Area Definition and Instance



To create a new Business Area instance:

1. In a Work Area, select **Business Area** from the **Add** drop-down list.
2. Click **Go**.

   The system displays the Create Business Area screen.
3. Choose one of the following options:
   - **Create a new Business Area definition and instance**. Choose this option if no Business Area definition exists that can meet your needs, either as it is or with some modification.
   - **Create an instance from an existing Business Area definition**. Choose this option if a Business Area definition already exists that meets your needs.

     If you can adapt an existing Business Area definition to make it fit your needs, first copy it into the current Application Area, then choose this option and select the copied definition. See Finding an Appropriate Definition and Reusing Existing Definitions for further information.

4. Depending on your choice, follow one of the following sets of instructions:

- Creating a New Business Area Definition and Instance
- Creating an Instance of an Existing Business Area Definition

Creating a New Business Area Definition and Instance

When you select **Create a new Business Area definition and instance** in the Create Business Area screen, additional fields appear.

1. Enter values in the following fields:
   - **Name**. See Naming Objects.

     > **Note:**
     >
     > For OBIEE Business Areas, the Business Area name is displayed in the OBIEE Answers user interface as the Subject Area.

   - **Description**. See Creating and Using Object Descriptions.
   - **Business Area Type**. Select the adapter from the drop-down list.

     If you select GENERIC_VISUALIZATION, you must then enter a schema name for the Business Area. See Defining Generic Visualization Business Areas for more information.
2. In the **Classification** section, select the following for both the definition and the instance:
   - **Subtype**. Select a subtype according to your company's policies.
   - **Classification Values**. See Classifying Objects and Outputs for instructions.
3. Click **Apply** to save your work and continue defining the Business Area.

   The system opens the Properties screen for the new Business Area instance.
4. Define the Business Area details. For information and instructions see:
   - Defining Table Descriptors
   - Defining Joins
   - Defining Business Area Hierarchies

   > **Note:**
   >
   > A Generic Visualization Business Area does not contain Joins, Hierarchies, Source Code, or Parameters.

5. Click **Check In**. The system checks in Version 1 of both the Business Area definition and instance.
6. Install the Business Area instance (see Using, Installing, and Cloning Work Areas). On creation of the Generic Visualization Business Area, its status is 'Not Installable'.

> **Note:**
>
> When installing a GVA Business Area instance, a database schema is created. The schema name is an attribute of the Generic Visualization Business Area instance and its default value is the Business Area instance name. The schema name must be unique within the Oracle LSH installation. If the name is not unique, Namings' uniqueness rule appends _**n** to the schema name; where **n** is the next non-unique integer. If a schema name is not unique, the installation fails.

7. Validate both the definition and the instance according to your company's policies.

Creating an Instance of an Existing Business Area Definition

If you use an existing Business Area as a definition source, its Table Descriptors, Joins and Hierarchies (if any) are already defined. See Creating an Instance of an Existing Definition for instructions.

After you have created the Business Area instance, you must map the Table Descriptors to Table instances; see Mapping Table Descriptors to Table Instances for instructions.

Using the Business Area Properties Screen

If you are working in a Work Area, you see the properties of both the Business Area instance and the Business Area definition it references. If you are working directly on the definition in an Application Area or Domain, you see only the properties of the definition.

See Modifying Business Areas for information on modifying Business Areas.

See also Figure 11-1.

This section contains the following topics:

- Instance Properties
- Definition Properties
- Business Area Attributes
- Buttons

Instance Properties

You can see the following instance properties:

- **Name:** You can click **Update** and modify the name. See Naming Objects for further information.
- **Description:** You can click **Update** and modify the description. See Creating and Using Object Descriptions for further information.
- **Definition:** This field specifies the Business Area definition to which this Business Area instance points.

You can upgrade to a new version of the same definition. See Upgrading to a Different Definition Version from an Instance.

- **Validation Status:** This field displays the current validation status of the Business Area instance. If you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Business Area: Installable or Non Installable. See Installation Requirements for Each Object Type.
- **Version:** This field displays the current version number of the Business Area instance.
- **Version Label:** This field displays the version label, if any, for the current Business Area instance version.
- **Schema Name:** (Generic Visualization Business Areas) The unique name of the schema created for the Business Area.
- **Default Blinding Access Type:** (Generic Visualization Business Areas) The data displayed by default in visualizations based on this Business Area.
- **Default Snapshot Label:** (Generic Visualization Business Areas) The data currency displayed by default in visualizations based on this Business Area.

  For further information on object versions, see Understanding Object Versions and Checkin/Checkout.

Definition Properties

- **Checked Out Status:** This field displays the status of the definition: either Checked Out or Checked In. You must check out the definition to modify Table Descriptors, Joins, and Business Area Hierarchies. However, you can change Table Descriptor mappings without checking out the definition. See Understanding Object Versions and Checkin/Checkout for further information.
- **Latest Version:** If set to **Yes**, this Business Area instance is pointing to the latest version of the Business Area definition. If set to **No**, this Business Area instance is pointing to an older version of the Business Area definition.
- **Checked Out By:** This field displays the username of the person who has the Business Area definition checked out. See Understanding Object Versions and Checkin/Checkout for further information.
- **Version Label:** This field displays the version label, if any, for this definition version.
- **Business Area/Adapter Type:** This field displays the name of the Business Area adapter.
- **Validation Status:** This field displays the current validation status of the Business Area definition. If you are working directly in the definition in an Application Area or Domain and you have the necessary privileges, you can change the validation status by selecting **Validation Supporting Information** from the **Actions** drop-down list. If you are working in an instance of the Business Area in a Work Area, and you want to change the validation status of the definition, you must go to the definition. See Validating Objects and Outputs for further information.
- **Status:** This field displays the installable status of the Business Area: Installable or Non Installable. See Installation Requirements for Each Object Type.

> **Note:**
>
> When a Generic Visualization Business Area Definition is created, its status is 'Not Installable'.

Business Area Attributes

This section of the Business Area screen displays attributes for the type of Business Area you have created. See Setting Business Area Attributes and Parameters for more information.

Buttons

From a Business Area instance in a Work Area, you can use the following buttons:

- **Install:** Click **Install** to install the Business Area instance, including mapped Table instances in the same Work Area; see Installing Business Area Instances. For a list of reasons a Business Area instance may not be installable, see Installation Requirements for Each Object Type.
- **Launch Visualization:** Click **Launch Visualization** to go to the Launch Visualization screen; see Launching Visualizations. You can see the **Launch Visualization** button only after you have installed the Business Area. See Installing Business Area Instances.
- **Update:** Click **Update** to modify the Business Area instance properties. See Modifying Business Area Instance Properties.
- **Check In/Out and Uncheck:** Click these buttons to check out, check in, or uncheck the Business Area definition. Different buttons are displayed in the Business Area Definition Properties section depending on the Checked Out Status and whether or not you are the person who has the definition checked out. If someone else has checked out the definition, you cannot check it in or uncheck it. The username of the person who has checked it out is displayed. See Understanding Object Versions and Checkin/Checkout.

Defining Table Descriptors

You make data available to visualizations by mapping the Table instances that contain the data to Table Descriptors in a Business Area. Any one visualization can access data through one and only one Business Area. The more Table Descriptors you include in a Business Area, the more data is accessible to a single visualization.

However, system performance during visualization creation improves by having fewer Table Descriptors in the Business Area. Therefore, you may want to define more Business Areas with many different combinations of Table Descriptors.

Business Areas have source Table Descriptors only. They can read data but they cannot write data to Tables instances.

If you do not want data from a particular Table instance Column available to visualizations, remove the Column from the Table Descriptor that is mapped to that Table instance.

You can map Business Area Table Descriptors to Table instances in any Work Area.

If the Table instances to which you want to map a Business Area's Table Descriptors already exist, the simplest way to add and map Table Descriptors to the Business Area is to use the **Table Descriptors from Existing Table Instances** job from the **Actions** drop-down list. See Creating Table Descriptors from Table Instances and Simultaneously Mapping Them for further information.

> **Note:**
>
> Set the **SAS Library Name** of the source Table Descriptor to `$REPINIT` if you want to enable queries from the OBIEE repository's Repository Init Block to access Table instances mapped to this Table Descriptor. These queries fetch data from the Oracle LSH Table instances at user-defined refresh intervals to initialize Repository Variables.
>
> If you do not set the **SAS Library Name** to `$REPINIT`, OBIEE cannot access the mapped Table instances and the automatic, periodic OBIEE repository queries will fail.

For additional information about Table Descriptors see the following sections:

- About Table Descriptors
- Creating a Table Descriptor
- Mapping Table Descriptors to Table Instances

Defining Joins

You can define a Join between two Table Descriptors to allow a visualization to query and display data from two Table instances as if they were one unified table.

> **Note:**
>
> Joins are not supported in Generic Visualization Business Areas.

You must define Table Descriptors before you can define Joins between them. You can create Joins between two and only two Table Descriptors.

You can define a Join between Table Descriptor A and Table Descriptor B, another Join between Table Descriptors B and C, and another between Table Descriptors C and D, for example. Furthermore, in this same example you can create a join between Table Descriptors D and A.

However, if you create a Join between Table Descriptor A and Table Descriptor B, you cannot create a Join between the same Table Descriptors with their positions reversed (Table Descriptor B and Table Descriptor A).

Joined Columns: For each Join, you must specify at least one pair of equivalent Columns, one Column from each Table Descriptor, that are equivalent in terms of their

data content; for example, Table A's Patient Column and Table B's Pat Column, both of which contain the patient ID. The joined Columns must be of the same data type.

> **Note:**
>
> For Business Areas of type OBIEE, joins work only if Table instances have a Primary Key constraint.
>
> Also, OBIEE Business Areas support only one-to-many relationships; for example, if you want to create a join between an employee table and a department table, you must add the department table to the Join first, because a department can have multiple employees in it, but an employee can belong to only one department.

If you specify more than one pair of equivalent Columns, the system sees rows in the two Table instances as being the same only if they have the same value in both Columns. For example, if you create a Column Join between Table A's Investigator Column and Table B's Inv Column, both of which contained an investigator ID, as well as the Patient/Pat join, the visualization query would see rows as being the same in the two Table instances only if they shared the same investigator ID as well as the same Patient ID.

> **Note:**
>
> Although some visualization tools allow the use of other operators to join columns, Oracle LSH always applies the "equal to" (=) operator.

Inner and Outer Joins: By default, an Oracle LSH Join is an inner join. That is, only rows where the joined Columns share the same value are evaluated for the query. Rows that exist in either Table instance that do not have an equivalent row in the other Table instance are not included in the visualization.

You have the option to create an outer join. In an outer join, the visualization query evaluates all the rows in the Table instance you define as Table A plus all the rows in Table B where each joined Column has the same value as a row in Table A.

- Defining a Join at the Table Level
- Defining a Join at the Column Level

Defining a Join at the Table Level

To define an Oracle LSH Join:

1. From the Joins subtab on the Properties screen of the Business Area, click **Add**. The system displays the Join for Business Area screen.
2. Enter a name for the Join.
3. Enter a description for the Join (optional).
4. Click the Search icon to the right of the Table A field.

5. Click **Go** to see a list of all the Table Descriptors you have defined for the Business Area. Alternatively, enter the exact name of the Table Descriptor you want, or use special characters if you are unsure of the name (see "Searching" in the *Oracle Life Sciences Data Hub User's Guide* for instructions).

   The system displays one or more Table Descriptors.

6. Click the Quick Select icon to select the Table Descriptor you want.

> **Note:**
>
> In an outer join, the system evaluates all the rows of the Table instance you map to the Table Descriptor you define as Table A.

7. If you want to define an outer join, select Yes in the **Table A Outer Join** field.
8. Click the Search icon to the right of the Table B field.
9. Click **Go** to see a list of all the Table Descriptors you have defined for the Business Area. Alternatively, enter the exact name of the Table Descriptor you want, or use special characters if you are unsure of the name (see "Searching" in the *Oracle Life Sciences Data Hub User's Guide* for instructions).

   The system displays one or more Table Descriptors.

10. Click the Quick Select icon to select the Table Descriptor you want.
11. Click **Apply**. See Defining a Join at the Column Level.

Defining a Join at the Column Level

To define which two Columns are joined, do the following:

1. Define the Table Descriptors in the Join; see Defining a Join at the Table Level.
2. In the Join properties screen, click **Update**. The system displays an Add Join Column button.
3. Click **Add Join Column**. The system displays a drop-down list under each Table Descriptor included in the Join.
4. For the Table Descriptor on the left, select the Column you want to define as part of the Column Join from the drop-down list.
5. For the Table Descriptor on the right, select the Column you want to define as part of the Column Join from the drop-down list.

> **Note:**
>
> The two Columns must have the same data type.

6. Click **Apply**. The system saves your work.

   Click Return to go back to the main Business Area screen.

Defining Business Area Hierarchies

Many visualization tools allow drilling down into more and more detailed data or up into more and more general data. To allow this functionality for visualizations of Oracle LSH data, you must define Business Area Hierarchies.

> **Note:**
>
> Hierarchies are not supported in Generic Visualization Business Areas.

Business Area Hierarchies define hierarchical relations among Table Columns to organize data in a meaningful way, from general to specific, so that data values at the lower levels aggregate into the data values at higher levels.

For example, you could define a Hierarchy for the Columns Study, Patient, Visit, Test, in that order, from the most general to the most specific. The system uses the Hierarchy and the data values for each row to interpret the data values in the Columns hierarchically: all tests belong to a visit; all visits are associated with a patient, and all patients are part of a study.

If Table Descriptors are defined as joined, you can create a Business Area Hierarchy that includes Columns from both Table Descriptors. For example, you could create a Business Area Hierarchy for the Columns Investigator Name, Patient, and Gender where the Investigator Name is in the Sites Table and the Patient and Gender Columns are in the Demography Table.

When you define a Business Area Hierarchy, you assign each Column an order number. The Column with order number one is the top, or most general, level of the Business Area Hierarchy, and each subsequently numbered Column is the next lower Hierarchy level. Sequential Columns must be contained either in the same Table Descriptor or in joined Table Descriptors.

- Grouping Columns

Grouping Columns

You can include two or more Columns in the same Business Area Hierarchy level by using the Group With Previous flag. For example, you could define the following Business Area Hierarchy:

The example above shows a three-level hierarchy. Investigator ID and Investigator Name are both in the top level, which makes sense because they are alternate values for the same type of information. Patient is the middle level and Gender is the bottom level. In a visualization, users can drill down from the investigator by either name or ID, or view them both, to all patients for whom a particular investigator is responsible, to an investigator's male patients or the same investigator's female patients.

To define a Business Area Hierarchy:

1. From the Business Area Hierarchies subtab on the Properties screen of the Business Area, click **Add**. The system displays the Create Business Area Hierarchies screen.

2. Enter a name for the Business Area Hierarchy (required).
3. Enter a description for the Business Area Hierarchy (optional).
4. Click **Apply**. The system displays the Properties screen of the new hierarchy.
5. Click **Update**. The system adds the Add Another Row button.
6. Click **Add Another Row**.
7. Click the Search icon in the **Table** column. The system opens a Search and Select screen.
8. Click **Go** to see a list of all the Table Descriptors you have defined for the Business Area. Alternatively, enter the exact name of the Table Descriptor you want, or use special characters if you are unsure of the name (see "Searching" in the *Oracle Life Sciences Data Hub User's Guide* for instructions).

   The system displays one or more Table Descriptors.
9. Click the Quick Select icon to select the Table Descriptor that is mapped to the Table instance you want to include in the Hierarchy.
10. Click the Search icon in the **Column Name** column. The system opens a Search and Select screen.
11. Click **Go** to see a list of all the Columns in the Table Descriptor. Alternatively, enter the exact name of the Column you want, or use special characters if you are unsure of the name (see "Searching" in the *Oracle Life Sciences Data Hub User's Guide* for instructions).

    The system displays one or more Columns.
12. Click the Quick Select icon to select the Column you want to include in the Hierarchy.
13. Repeat Steps 6-12.

    This time the system displays only the Table Descriptor you selected in Step 9 and any other Table Descriptors that are part of a Join that includes the Table Descriptor you selected.

    In addition, beginning with the second row, you can choose to group the Column with the previous one at the same level of the Hierarchy; see Grouping Columns.
14. When you have added all the rows you want in the Hierarchy, click **Apply**. the system saves your changes.

    Click **Return** to go back to the Business Area Properties screen.

Example 11-1 Business Area Hierarchy Column Grouping

Order number	Column	Group With Previous?
1	Investigator ID	—
2	Investigator Name	Yes
3	Patient	—
4	Gender	—

Understanding Business Area Source Code

In the context of an Oracle LSH Business Area, the Source Code object is used to store files required for integration with an external visualization tool. This is different from the way Source Code is used in Oracle LSH Programs where the Oracle LSH Source Code object contains actual program source code.

For information on OBIEE Business Area Source Code see About OBIEE Business Areas.

> **Note:**
>
> Source Code is not supported in Generic Visualization Business Areas.

Setting Business Area Attributes and Parameters

Some types of Business Areas, including third-party types not covered by this documentation, may have attributes and Parameters. When you create a Business Area of those types, attributes or Parameters required for the visualization tool may appear on the Business Area Properties page.

OBIEE Business Areas have one attribute; see Defining Oracle Business Intelligence Business Areas.

Defining Oracle Business Intelligence Business Areas

For information on using the OBIEE Administration Tool and Presentation Services to create visualizations—Answers and Dashboards—see the Oracle Business Intelligence documentation, including online help.

This section contains the following:

- About OBIEE Business Areas
- Defining an OBIEE Business Area
- Manually Deploying the Master RPD File
- Starting the WebLogic Server
- Visualizing Business Area Data using OBIEE Answers
- OBIEE Security
- Installing and Setting Up Oracle Business Intelligence Administration Tool

About OBIEE Business Areas

When you install an OBIEE Business Area for the first time the system generates an Oracle BI repository (.rpd) file—which is required for creating and displaying Subject Areas, Answers, and Dashboards in Oracle BI Presentation Services—and copies it to the BI Server. This generating .rpd allows the creation of visualizations of LSH data

through OBIEE Answers without the use of the OBIEE Administration tool. Oracle LSH stores the .rpd file in the OBIEE Business Area Source Code definition.

The system deploys the .rpd files from all the OBIEE Business Areas that share the same OBIEE Service Location Name onto that Oracle BI Server and merges them into a single .rpd file as each Business Area is installed.

If you are using OBIEE 11g, each time you install an OBIEE Business Area you must open Oracle Enterprise Manager, manually deploy the merged, or master, .rpd file and restart the BI Service for the Business Area to become available to users; see Manually Deploying the Master RPD File. If you are using OBIEE 10g, this process is automatic each time you install the Business Area.

Customizing the RPD File Using OBIEE Administration Tool: Oracle LSH takes advantage of only a small number of the features that OBIEE has to offer. If you are experienced with OBIEE you may wish to make changes to the Business Area's default .rpd file using the OBIEE Administration Tool and upload the resultant .rpd to replace the system-generated .rpd in the Business Area in Oracle LSH.

In order to ensure that subsequent installations of the Business Area do not overwrite those customizations, the system tracks the origin of the .rpd file. When Oracle LSH creates an .rpd file during installation, it populates the File Reference Name with the value `SYSTEM`. When a user uploads a customized .rpd file the system populates the File Reference Name with the value `USER`.

Each time you install the Business Area, the system checks if the current .rpd file was generated by the system or manually uploaded to the Business Area. If the file was generated by Oracle LSH and not manually uploaded and the Table Descriptor, Join, or Hierarchy definitions have changed, then the system generates a new .rpd file and stores this as a new version of that Source Code.

If the current .rpd file was uploaded, the system does not override it with a new file even if you have made changes in Oracle LSH. You must make corresponding changes—adding, deleting, or modifying tables, joins, and hierarchies—in the Administration Tool and reupload the file.

Defining an OBIEE Business Area

Oracle recommends developing an OBIEE Business Area in the following order:

1. Create the Business Area; see Creating a Business Area. Special information for OBIEE Business Areas:
    - **Name**. The Business Area name is displayed in OBIEE Answers as the Subject Area. **The name must be unique among Business Areas using the same service location**. If it is not unique, when you install the Business Area you receive an error message with the location of the previously installed Business Area with the same name.

      > **Note:**
      >
      > Do not change the name of a previously installed Business Area. This will cause problems with the .rpd file.

- **OBIEE Service Location Name**. Select the value for the computer with the Presentation Server you want this Business Area to use, if more than one is available in your company.

2. Add Table Descriptors and map them to the Table instances that contain the data you want to make available in OBIEE; see Defining Table Descriptors.

> **Note:**
>
> To enable the OBIEE repository to query the source Table instances that the Oracle LSH OBIEE Business Area uses, set the **SAS Library Name** of the mapped Table Descriptor to `$REPINIT`. See Defining Table Descriptors.

3. If you want to add joins and hierarchies in Oracle LSH, do so now; see Defining Joins and Defining Business Area Hierarchies. See About OBIEE Business Areas.

> **Note:**
>
> Do not explicitly create a Source Code definition. The Source Code is automatically created when you install the Business Area. Its name is determined by the value of the Details attribute in the Deploy service defined for the service location you selected.

4. Install the Business Area to create the initial .rpd file. Install it again if you make changes in Oracle LSH before working in the OBIEE Administration Tool. The system generates a new .rpd file with your changes.

> **Note:**
>
> Customization in the OBIEE Administration Tool is optional.

5. Click **Launch IDE** to open the OBIEE Administration Tool. When prompted for a repository password, enter blank and continue. Work on the .rpd file in this tool, using the OBIEE user documentation.

   If you make structural changes in the OBIEE Administration Tool—changes to tables, joins, or hierarchies—you must make the same changes in Oracle LSH so that the .rpd file is always synchronized with the Business Area definition. Otherwise you will not be able to use the new structures in OBIEE. For example, if you add a column to a table in OBIEE but do not add the column in the Business Area Table Descriptor and map it to a column in its Table instance, you will not be able to see the data in the column in OBIEE.

6. When you are finished making changes, upload the .rpd file to the Business Area:

   a. In the Source Code tab, click the link to the Source Code definition.

   b. Click **Browse...**. The .rpd file is generally stored on the C drive, in the CdrWork folder. The full path of the repository file looks like this:

```
Drive:\CdrWork\LSH_database_username\OBIEE_BA_Domain_name
\OBIEE_BA_Application_Area_name\OBIEE_BA_Work_Area_name
\OBIEE_BA_name\OBIEE_BA_version_number\
```

> **Note:**
>
> Oracle LSH supports uploading zipped .rpd files. However, you must give the .zip file exactly the same name as the .rpd file; for example, `test.zip` must contain `test.rpd`.

c. Click **Apply**.

7. Install the Business Area. If you are using OBIEE 10g, the system creates a new version of the Source Code definition that contains the newly uploaded file and deploys the file to the repository folder path defined on the Oracle BI Server, merging it into the master .rpd file.

> **Note:**
>
> If an .rpd file does not appear in the Source Code tab after installation, check the Jobs subtab. If the Execution Status is Pending Logon or Obtain Service, there is a problem with the Distributed Processing (DP) Server on the BI Server computer.

8. If you are using OBIEE 11g, manually deploy the master .rpd file to the BI Server and restart the BI Service; see Manually Deploying the Master RPD File.

Manually Deploying the Master RPD File

Manually deploy the new version of the master .rpd file on your OBIEE 11g BI Server using the Oracle Enterprise Manager. This step is required every time you install a Business Area.

1. Open the Oracle Enterprise Manager using the URL specific to your environment.

> **Note:**
>
> If the URL does not work, you may need to restart the WebLogic Server; see Starting the WebLogic Server.

2. In the left-hand panel, navigate to Farm_bifoundation_domain, then Business Intelligence, then coreapplication. Then click the **Deployment** tab, and then the **Repository** subtab.
3. Click **Lock and Edit Configuration** near the top. A confirmation message appears.
4. Under Upload BI Server Repository, click the **Browse** button for the Repository File field and select the master .rpd file.

> **Note:**
>
> The location of the master .rpd file is determined by the edited obieedeploy.cmd file; ask your administrator for it.

5. Enter the repository password and confirm password. The password **must** be same as the administrator password stored in LSH under Remote Location Connections. This is very important for the integration. Click **Apply**.
6. Click **Activate Changes** at the top of the screen. A confirmation message appears.
7. Click **Restart** and confirm.
8. Ensure that Restart All completed successfully. This indicates the successful deployment of the .rpd on the BI Server and a successful restart of the BI Server services. The Business Area .rpd is now ready to be used through the OBIEE Presentation Service (BI Answers).

Starting the WebLogic Server

If the URL for Oracle Enterprise Manager is not working, the WebLogic Server may be down. To start it:

1. On the BI Server computer, right-click **Command Prompt** under the Start menu and select **Run as Administrator**.
2. Change directory to the Domain folder under the OBIEE installation folder; for example, E:\oracle\fmw\user_projects\domains\bifoundation_domain.
3. Run the command `startWeblogic.cmd` under this folder. The system prompts you for the WebLogic Server username and password.
4. Check that the command window displays the message "Server started in RUNNING mode."

Visualizing Business Area Data using OBIEE Answers

You can access OBIEE data visualizations in two ways:

- **In Oracle LSH**, go to the Visualizations subtab of the Reports tab; locate the Business Area under its classification hierarchy; click the View icon in the Actions column for the Business Area; and click **Launch Visualization**.

  The system launches the Oracle Business Intelligence Presentation Services interface. Navigate to the Subject Area with the same name as the Business Area and follow instructions in the OBIEE user documentation to create Answers and Dashboards.
- **Outside Oracle LSH**, go to the URL of the computer corresponding to the Service Location Name. The system launches the Oracle Business Intelligence Presentation Services interface. Navigate to the Subject Area with the same name as the Business Area and follow instructions in the OBIEE user documentation to create Answers and Dashboards.

OBIEE Security

OBIEE security is handled through Oracle LSH.

- End User Security
- Developer Security

End User Security

Security access to OBIEE visualizations is determined by users' security privileges on the Business Area on which the visualization is based. The automatically generated RPDUpdates.txt file contains the information required to initiate user authentication. This file is passed to the BI Server during Business Area installation and RPD file merge and deployment operation.

When you launch an OBIEE visualization from within Oracle LSH , you can see noncurrent and blinded Oracle LSH data if you have the required privileges.

If you access an OBIEE visualization outside of Oracle LSH, through the URL of the OBIEE Presentation Services, you cannot see blinded or noncurrent data regardless of your privileges.

Developer Security

Developers must have normal security access to modify the Business Area.

When you launch the OBIEE Administration Tool or an OBIEE visualization from within Oracle LSH , you can see noncurrent and blinded Oracle LSH data if you have the required privileges.

When you launch the OBIEE Administration Tool from Oracle LSH, the system create synonyms to the Business Area's Table Descriptors for your database account, and the OBIEE Administration Tool connects to the LSH database with your database account. The logon trigger authenticates against Oracle LSH user privileges.

Installing and Setting Up Oracle Business Intelligence Administration Tool

To use the Oracle Business Intelligence Administration Tool, you must do the following on your local PC:

- Get the CD-ROM that contains the Oracle LSH files **cdrconfig.xml** and **cdrclient.exe** from your system administrator and insert it into your PC. InstallShield automatically runs setup.exe to load cdrconfig.xml and cdrclient.exe to a location you specify on your local computer.
- Install Oracle BI Client Tools on your PC in the location specified by your system administrator. The location must match the directory path specified in **cdrconfig.xml**.
- Ensure that **cdrconfig.xml** has the correct directory path for the Oracle BI Administration tool.
- In the Oracle BI Administration Tool, set up connectivity with the Oracle LSH database. See Set Up Database Connectivity for information.

- Add and set the following environment variables on the PC through advanced system properties:
  - **ORACLE_HOME** Enter the absolute path of the Oracle Home on your PC; for example:

    ```
    Drive:\Program Files\Oracle Business Intelligence Enterprise Edition Plus Client\oraclebi\orahome
    ```

  - **ORACLE_INSTANCE** Enter the absolute path to Oracle BI orainst1 on your PC; for example:

    ```
    Drive:\Program Files\Oracle Business Intelligence Enterprise Edition Plus Client\oraclebi\orainst
    ```

  - **ORACLE_BI_APPLICATION** You must enter the following value exactly:

    ```
    coreapplication
    ```

- Set Up Database Connectivity

Set Up Database Connectivity

To access Oracle LSH data in the Administration Tool on your PC, you must set up connectivity between the Oracle LSH database and your PC. You can use a system ODBC DSN or the Oracle Call Interface (OCI10g):

- **Set Up Database Connectivity Using a System DSN**
  1. Create a valid Oracle driver-based System Data Source Name (DSN) on your PC that points to the LSH database.
  2. Consult Microsoft Windows online help for instructions on creating the System DSN.
  3. Use the Test Connection button to test the connection.
- **Set Up Database Connectivity Using OCI 11g**
  1. Create an entry in your PC's tnsnames.ora file with the name and connection details for the LSH database server.
  2. Create an entry in your PC's tnsnames.ora file with the name and connection details for the LSH database server.
  3. Test the connection by trying to connect to the Oracle LSH database using sqlplus as apps. For example:

     ```
     sqlplus apps/apps_password@lsh_database_name
     ```

Defining Generic Visualization Business Areas

Oracle LSH includes a Generic Visualization adapter to allow you to integrate the data visualization tool of your choice with Oracle LSH so that your end users can view Oracle LSH data through that tool. See the *Oracle Life Sciences Data Hub Adapter Toolkit Guide* for information on using the Generic Visualization adapter.

After setting up integration, you define and install one or more Business Area instances of type GENERIC_VISUALIZATION. Users can then access the data visualizations through the tool's URL. Users can then access data visualizations through the tool's URL.

- Generic Visualization Business Area Instance Properties
- Assigning Security Privileges to Business Area Data

Generic Visualization Business Area Instance Properties

Generic Visualization Business Area instances have the following properties:

- **Schema Name:** Enter a unique name for the schema of up to 30 characters. The value is stored in uppercase. Do not use special characters or reserved words; see Naming Objects. Users will be able to use this account to log in to the integrated visualization tool.

  If a schema already exists in the Oracle LSH installation with the same name, the system automatically appends _*n* to the name, where *n* is 1 or the next higher integer if there is already a schema name *your_name*_1 or higher.

- **Default Currency:** This setting controls the data viewed by users if they do not explicitly make another selection at login. The default value is Current. If all the source Table instances mapped to the Business Area have the same snapshot label applied and you want users to see that snapshot data by default, select the snapshot label.

- **Default Blinding Access Type:** This setting controls the data available to users. Users' privileges also determine which data they can view. Your own privileges determine which values you can set. The possible settings include:

  – **NA/Dummy** All users, regardless of their privileges, can see data in Table instances whose Blinding flag is set to **No** and the dummy data in Table instances whose Blinding flag is set to **Yes**. If all Table instances mapped to Business Area Table Descriptors have their Blinding flag set to **No**, this is the only option available.

  – **Real(Unblinded)** If all Table instances whose Blinding flag is set to **Yes** have a Blinding status of **Unblinded**, you can set the Default Blinding Access Type to this value. In these Table instances, users with the necessary privileges can see the real, unblinded data, and users without these privileges see the dummy data. All users can see data in Table instances whose Blinding flag is set to **No**.

  > **Note:**
  >
  > The adapter can allow users with Blind Break privileges on all Table instances whose Blinding flag is set to **Yes** and whose Blinding Status is set to **Blinded** to select the **Real (Blind Break)** option when they log in, regardless of the Default Blinding Access Type. All blind breaks are audited.

- **Table Descriptors:** Generic Visualization Business Area Table Descriptors can be mapped to any Table instance type including 'View' Table instances.

> **Note:**
>
> You cannot define Joins, Hierarchies, Parameters, or Source Code for Generic Visualization Business Areas.

Assigning Security Privileges to Business Area Data

Unlike other Business Areas, which are installed in their Work Area's schema, Oracle LSH installs each Generic Visualization Business Area instance in its own schema outside the Work Area schema. There are simplified security requirements for data in this schema.

Users can log in to the integrated visualization tool using an Oracle LSH database account. The system checks if there is an Oracle LSH user account linked to the database account. If there is a linked user account, the system uses it to determine the user's privileges. If there is no linked user account, the system uses the database account itself to determine the user's privileges.

The database account can have one or two privileges assigned:

- **Read Data**. This privilege allows the user to view data that was never blinded and dummy data in Table instances that are currently blinded. All database accounts that should have access to the Business Area instance data should have this privilege.
- **Read Unblind**. This privilege allows the user to view data that has been permanently unblinded.

If a user should be able to view currently blinded data, he or she must have an Oracle LSH user account with all the required Blind Break privileges and a linked database account. An administrator must set up the user account and appropriate privileges.

If you have the Manage GVA BA Database Access operation on Business Area instances and belong to a user group assigned to a Business Area instance, you can grant or revoke Read Data and Read Unblind privileges database accounts for the Business Area instance. Oracle LSH audits all changes to these permissions.

To grant privileges to database accounts on Business Area instance data:

1. In the Business Area instance Properties page, select **Manage DB Privileges** from the Actions drop-down list. Select **Go**.

   You see the the privileges **Read Access** and **Read Unblind Access**. You can expand either privilege to view the database accounts currently assigned to that privilege.
2. To change assignments, select the plus (**+**) icon in the **Manage** column for either Read Access or Read Unblind Access. A screen opens displaying available accounts for assignment and those already selected for the privilege.
3. Use the arrow icons to grant or revoke the selected privilege to one or more accounts:
   - To grant an account the privilege you selected, double-click or use the arrow icons to move the account from Available Users to Selected Users.
   - To revoke the privilege, double-click or use the arrow icons to move an account from Selected Users to Available Users.

4. Click **Apply**.

Launching Visualizations

This section contains the following topics:

- Creating a Visualization
- Setting Data Currency and Blinding Values

Creating a Visualization

To create a Visualization from a Business Area instance:

1. Click **Launch**. Oracle LSH opens the Launch Visualization screen.

> **Note:**
>
> Launching the IDE is not supported for Generic Visualization Business Areas.

2. The Launch Visualization screen displays details about the Business Area instance, and launch settings—data currency, and data blinding-related settings—for data that becomes available to OBIEE. To edit these settings, click **Launch Settings**. Oracle LSH opens the Launch Settings screen. See Setting Data Currency and Blinding Values.
3. After making your selections in the Launch Settings screen, click **Launch** on the Launch Visualization screen. Oracle LSH opens the OBIEE Administration Tool.

> **Note:**
>
> When you launch the OBIEE Administration Tool, you have to log in with an Oracle LSH username and password.

Setting Data Currency and Blinding Values

You can determine the blinding status and currency of the data you see by clicking the Launch Settings button and selecting a value for Blind Break and Shared Snapshot Label.

This section contains the following topics:

- Setting the Blind Break Value
- Setting the Shared Snapshot Value

Setting the Blind Break Value

This setting is relevant only when one or more source Table instances either currently or formerly contained blinded data. Special privileges are required to view real data

that is either currently blinded or was formerly blinded. You must have these special privileges on all such Tables, in order to see real data in any of them.

> **Note:**
>
> You must have Read Data privileges in order to see any data at all.

The following choices are available depending both on the state of the data and on your security privileges:

- **Not Applicable**. If none of the data has ever been blinded, the only option available is **Not Applicable**. No special privileges are required.
- **Dummy**. This is the only option available to you if you do not have blinding-related privileges for blinded Tables. You can also see this option if you have blinding-related privileges. In that case, you can select this option to work with dummy (not real) data in Oracle Business Intelligence Enterprise Edition.
- **Real (Blind Break)**. If any of the data is currently blinded, and you have the required privileges, you can select this option to view real data in Oracle Business Intelligence Enterprise Edition according to your company's policies. The table(s) containing the blinded data remain blinded after you run the visualization.
- **Real (Unblinded)**. If a blinded Table instance has now been unblinded, you can see real data for the Table instance, provided you have the required privileges. If there are more than one such Table instances, you need the required privileges for all of them to be able to use this option.

Setting the Shared Snapshot Value

If all the relevant Table instances share one or more snapshot labels, those snapshot labels appear in this drop-down list and you can select one.

> **Note:**
>
> If a Table instance is defined as a pass-through view, visualizations always display current data in it, regardless of this setting.

In addition, you normally have the option to view the current data.

You can apply snapshot labels in the Work Area; see Adding, Removing, or Moving a Snapshot Label.

For more information on what snapshots are, see Data Snapshots.

Installing Business Area Instances

You can install a Business Area instance directly from its Properties screen, using the Install button, or in its Work Area (see Installing a Work Area and Its Objects).

When you install a Business Area instance using the **Install** button on its Properties screen:

- The system checks in the Business Area instance and definition, and also the Table instances in the current Work Area to which the instance is mapped.
- The system attempts to install the Business Area instance and its source Table instances in the current Work Area. The system displays a success or error message. If the installation fails, the error message displays the name of any objects that were not installable.

> **Note:**
>
> If the Business Area definition or any of its source Table instances in the current Work Area is not installable, the system cannot install the Business Area instance. See Installation Requirements for Each Object Type for the reasons these objects may not be installable.

- In an OBIEE Business Area, the system may generate a repository file; see About OBIEE Business Areas.

> **Note:**
>
> To continue working on the Business Area, check it out.

- Accessing the Log File

Accessing the Log File

To see the log file for the installation, you must go to the Work Area Installation screen, as follows:

1. Click the **Applications** tab. The main Application Development screen opens.
2. Click the name of the Work Area you are working in. The Work Area screen opens.
3. From the **Actions** drop-down list, select **Installation History**.
4. Click **Go**. The system displays the Installation History screen with the log files in chronological order.
5. Click the **View Log** link for the most recent installation attempt or for the date and time that you ran the install process. The system displays the log file.

For information on installation and on reading the log file, see Installing a Work Area and Its Objects.

Modifying Business Areas

If you have the necessary privileges, you can modify a Business Area either through an instance of it in a Work Area or directly in the definition in its Domain or Application Area. In most cases it makes sense to work through an instance in a Work Area for the following reasons:

- In order to use or test changes to the definition you must create and install an instance of it.
- If you work through an instance, the system automatically repoints the instance to the new version of the definition.

However, if you need to change properties of the definition, you must work directly in the definition in its Domain or Work Area.

Whether you work in an instance or directly in the definition, when you check in the new version of the definition you have the opportunity to upgrade instances of the original definition to the new version; see Upgrading Object Instances to a New Definition Version.

This section contains the following topics:

- Modifying Business Area Instance Properties
- Modifying Business Area Definition Properties

Modifying Business Area Instance Properties

On the Business Area instance's Properties screen, click **Update** to enter changes. Oracle LSH creates a new version of the instance you are working on and applies your changes to it when you click **Apply**. Click **Cancel** to discard your changes and the new version.

You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.

> **Note:**
>
> You must reinstall the Business Area for the changes to take effect.

You can modify the following:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.
- **Definition Source:** This field applies to the instance only. It specifies the Business Area definition to which this Business Area instance points. It generally does not make sense to change the source definition for the following reasons:
  - Changing the definition may result in a new set of Table Descriptors, Joins, and Business Area Hierarchies.
  - Any new Table Descriptors are not mapped.
  - The Business Area's status changes to **Non Installable**.

  If you want to change to a new version of the same definition, use the **Upgrade Instance** option from the **Actions** drop-down list.
- **Default Blinding Access Type:** (Generic Visualization Business Areas).
- **Default Currency:** (Generic Visualization Business Areas).

Modifying Business Area Definition Properties

You can go to a Business Area definition's Properties screen in one of the following ways:

- **From the Business Area's Properties screen**: Click the hyperlink of the Business Area definition that appears in the Definition field.
- **From the Domain or Application Area where you created the definition**: Click Manage Definitions to view all the definitions in that Domain or Application Area. Click the definition name.

Once on the Business Area definition screen, click **Update** to enter changes. Oracle LSH creates a new version of the definition. You can change the following properties:

- **Name:** See Naming Objects for further information.
- **Description:** See Creating and Using Object Descriptions for further information.

  You can modify some properties through the **Actions** drop-down list; see Using the Actions Drop-Down List for further information.
- Modifying Table Descriptors
- Modifying Joins
- Modifying Business Area Hierarchies
- Modifying Business Area Source Code

Modifying Table Descriptors

Table Descriptors belong to the Business Area definition, but Table Descriptor mappings belong to the Business Area instance. You must check out the definition to add, remove, or update Table Descriptors, but not to map, unmap, or remap Table Descriptors.

You can remove the existing Table Descriptors and add different ones. See Creating a Business Area for information about the function of Table Descriptors in a Business Area.

Oracle LSH enforces the following rules:

- You cannot remove a Table Descriptor if any Join or Hierarchy references it.
- You cannot change a Table Descriptor's source Table definition to a different Table definition if any Join or Hierarchy references the Table descriptor.
- You cannot remove a Column from a Table Descriptor if any Join or Hierarchy references it.

You can change the Table Descriptor mappings, which are part of the Business Area instance, not the definition; see Mapping Table Descriptors to Table Instances .

> **Note:**
>
> Oracle LSH does not prevent all changes that might cause your Business Area to not function properly. For example, you are allowed to change the underlying Variable of a Column even if the Column is part of a Join. However, if you select a Variable that has a different data type, the Join may no longer work.

Modifying Joins

Joins belong to the Business Area definition.You must check out the definition to add, remove, and modify Joins. See Defining Joins for information about Joins.

Modifying Business Area Hierarchies

Business Area Hierarchies belong to the Business Area definition. You must check out the definition to add, delete, and modify Business Area Hierarchies. See Defining Business Area Hierarchies for information about Business Area Hierarchies.

To change the order of the hierarchy columns, click Reorder. See Reordering and Renumbering Objects.

Modifying Business Area Source Code

Business Area Source Code belongs to the Business Area definition. You must check out the definition to make any changes to the Source Code instance, for example, editing and uploading an OBIEE repository file for OBIEE Business Areas. See About OBIEE Business Areas for more information on OBIEE Business Area Source Code.

12

Using, Installing, and Cloning Work Areas

This section contains information on the following topics

- Using the Work Area Properties Screen
- Personalizing Your Work Area Properties Screen
- Installing a Work Area and Its Objects
- Cloning Work Areas for Testing and Production

Using the Work Area Properties Screen

To reach the Work Area Properties screen, click the Applications tab and navigate to the Application Area that contains the Work Area, then click the Work Area's hyperlink. You can also use the Search field from many other Oracle LSH screens.

You can see information about the Work Area as a whole and about all the object instances it contains.

Oracle Life Sciences Data Hub (Oracle LSH) Work Areas are designed to allow many people to work simultaneously on developing different parts of a single application. In addition, depending on your company's policies, you may have one or more Work Areas of your own to work in. You may or may not be able to create Work Areas yourself.

As you define the objects required for a particular application, you must install them in the database before you can run the executable objects or write data to tables; see Installing a Work Area and Its Objects.

- Work Area Properties
- Viewing a Work Area's Installation History
- Viewing a Work Area's Version History
- Changing a Work Area's Usage Intent
- Object Instance Information
- Object Instance Actions
- Adding Object Instances to a Work Area
- Managing Table Instance Snapshot Labels in a Work Area

Work Area Properties

A Work Area itself has the following properties:

Usage Intent

Work Areas have a special property called Usage Intent that interacts with the validation status of the Work Area and its contained object instances to enforce certain

system behavior. The valid usage intent values are the same as the validation status, except for Retired. They are Development, Quality Control, and Production.

For an explanation of the interaction of usage intent and validation status, see "Work Area Usage Intent and Validation Status" in the *Oracle Life Sciences Data Hub Implementation Guide*.

Validation Status

The validation status of the Work Area itself. You cannot promote a Work Area to a higher validation status until every object it contains has a validation status equal to or greater than the new Work Area validation status.

Status

A Work Area's status is either Installable or Non Installable. The status is Installable only if every object instance contained in it is Installable. However, even if the status is Non Installable, you can run the installation job as follows:

- You can run upgrade or partial installation if you omit the noninstallable objects.
- You can run full installation. If an executable object is noninstallable, the system automatically performs automatic mapping by name for any unmapped target Table Descriptors, looking for Table instances of the same name in the current Work Area. If any target Table Descriptors remain unmapped, the system creates Table instances from them in the current Work Area and maps them. The executable object may then be installable.
- You can install individual objects; see Installing Individual Objects.

> **Note:**
>
> Even if a Work Area's status is Installable, you cannot install it if the source definition of any of its object instances is checked out by another user.
>
> However, if you have the Checkin Administrator privileges you can check in all objects that are checked out by other users and install the Work Area. The system displays a warning indicating that you are checking in objects checked out by other users. Normal object security privileges on the Work Area and its objects is also required.

Version

The current version of the Work Area. (You can see information about previous versions by selecting **View Version History** from the **Actions** drop-down list.)

Cloning Label

If this Work Area was the source or target of a Work Area clone, the system displays the label entered at the time of clone.

Version Label

If a user has applied a label to this version of the Work Area, the system displays it here. To create a label for the current Work Area version, select **Version Label** from the **Actions** drop-down list.

Domain

The system displays the Domain in which the Work Area is located.

Application Area

The system displays the Application Area in which the Work Area is located.

Forward Chain Enabled

If checked, Programs and other executables in the Work Area can be part of a forward chaining execution process if they are set up to do so. If unchecked, no Program or other executable in the Work Area can be part of a forward chaining process even if they are set up for it; see Forward Chaining for more information.

Viewing a Work Area's Installation History

To see a record of each previous installation of the Work Area, do the following:

1. From the **Actions** drop-down, select **Installation History**.
2. Click **Go**. The system displays the Work Area Install History screen.

The Install History screen includes the following information:

Installation Attempt

The system assigns a number *x*.*y* to each installation attempt, where *x* is the version number and *y* is installation attempt on that version represented by the row. For example, if the current Work Area version number is 3 and there have been 5 installation attempts on version 3, then the most recent installation attempt number is 3.5, and the previous one is 3.4.

Install Status

The system displays the final status of the installation attempt. If the installation was successful, the status is Installed. If the installation was not successful, the system displays the status corresponding to the last installation phase completed successfully. See Work Area Installation Phases and Statuses.

Install Mode

The system displays the mode used to run the installation: Full, Partial, or Upgrade. See Installation Modes for further information.

Force to Regenerate Scripts

If set to Yes, the installation process regenerated installation scripts even for objects that had not changed since the previous successful installation. If set to No, the installation process generated installation scripts only for objects that had changed since the last successful installation.

Installed By

The username of the person who initiated the installation attempt.

Installation Date

The date on which the installation was completed.

View Log

Click the icon in the **View Log** column to see the log file for the installation. See Reading the Log File for further information.

Viewing a Work Area's Version History

To see the Work Area's version history, do the following:

1. From the **Actions** drop-down, select **View Version History**.
2. Click **Go**. The system displays the Work Area's Version History screen.

The Version History screen includes the following information:

Name

The Work Area version's name.

Description

The Work Area version's description.

Version

The Work Area version's version number.

Status

The installation status: either Installable or Non Installable.

Validation Status

The validation status of the Work Area: either Development, Quality Control, or Production.

Usage Intent

The usage intent of the Work Area: either Development, Quality Control, or Production.

Last Modified By

The username of the person who last made any changes to the Work Area, including checking out any object instance in the Work Area.

Last Modified

The date of the last modification to the Work Area.

- Version Details

Version Details

To view additional details about a Work Area version, click the **Show** hyperlink or its plus (**+**) icon. The system displays the following information, if it exists:

Cloned From

If the Work Area version was created by cloning another Work Area onto this one, the system displays the name of the original Work Area.

Cloning Label

If the Work Area version was created by cloning another Work Area onto this one, the system displays the label created for the cloning operation.

Version Label

If a user created a label for the Work Area version, the system displays the label.

Changing a Work Area's Usage Intent

Work Areas have a special property called Usage Intent that interacts with the validation status of the Work Area and its contained object instances to enforce certain system behavior. The valid usage intent values are the same as the validation status, except for Retired. They are Development, Quality Control, and Production.

For an explanation of the interaction of usage intent and validation status, see "Work Area Usage Intent and Validation Status" in the *Oracle Life Sciences Data Hub Implementation Guide*. For a discussion of Work Area usage, see "Work Areas" in *Oracle Life Sciences Data Hub Implementation Guide*.

When a Work Area is first created, its usage intent is set to Development.

Special privileges are required to modify a Work Area's usage intent.

To modify the usage intent, do the following:

1. From the **Actions** drop-down list, select Update Usage Intent.
2. Click **Go**.
3. In the **Description** field, enter the reason you are making the change.
4. From the **Usage Intent** drop-down, select the value you want to apply: Development, Quality Control, or Production.
5. Click **Apply**. The system makes the change and returns you to the Work Area's Properties screen.

Object Instance Information

The Work Area properties screen can display the following information about each object instance contained in the Work Area. You can reduce the number of columns displayed and change the right-to-left order in which they are displayed by clicking the **Customize** button or selecting a different view; see Personalizing Your Work Area Properties Screen.

You can change the top-to-bottom display order of a Work Area's objects by sorting on the fields with an asterisk (*) below.

Name*

The name of the object instance. The name is hyperlinked; click it and the system opens the Properties screen for that object instance.

Type*

The object type.

Technology*

For executable object instances, this is the technology type; for example, **SAS** or **PLSQL** for Programs; **Oracle Clinical Labs** or **SAS** for Load Sets. For Table instances, it is the processing type; for example **Staging** or **Transactional with Audit**.

Description

The object description entered by its Definer.

Latest Version

The number of the latest version of the object instance.

Installed Version

The number of the version of the object instance that is currently installed.

Status*

Objects can have the following statuses:

- **Installable**. The object either is installed or can be installed.
- **Non Installable**. The object has problems that prevent it from being installed. See Installation Requirements for Each Object Type for the reasons each object type may be noninstallable.

> **Note:**
>
> The Install option for single object instances is always available, even for objects with a status of Non Installable. The system automatically addresses the problem of unmapped Table Descriptors during installation; see Installing Individual Objects.

- **Upgradable**. This status applies only to Table instances. If a Table instance is Upgradable, you can install it using any mode. If it is Non Upgradable, you cannot install it in upgrade mode. See information on Upgrade mode under Installation Modes.

> **Note:**
>
> An object instance of any type is not installable if its source definition is checked out by a user different from the person initiating the installation (except a Checkin Administrator), or if its source definition is not installable for any other reason.

Validation Status*

Development, Quality Control, Production, or Retired; See Validating Objects and Outputs for further information.

Has Data*

This field applies only to Table instances. If set to **Yes**, the Table instance contains data. If set to **No**, the Table instance does not contain data.

Definition Checked Out By*

If the source definition of the object instance is checked out, the system displays the username of the person who has checked it out.

Created TS *

Timestamp when the object was created.

Created By*

User ID of the person who created the object.

Last Modified TS*

Timestamp of the most recent modification of the object.

Last Modified By*

User ID of the person who most recently modified the object.

Installed TS*

Timestamp of the most recent installation of the object.

Last Submit TS*

This field applies only to executable instances: timestamp of the most recent submission of the object.

Last Refresh TS*

This field applies only to Table instances: timestamp of the most recent job that wrote data to the Table instance.

Browse Data, Install, Launch, Submit

See Object Instance Actions below.

Object Instance Actions

You can take actions on individual object instances by one or more of the following methods:

- Using the Icons in the Object's Row
- Using the Drop-Down List

Using the Icons in the Object's Row

The following actions are available on each object's row if your view displays them:

Browse Data

This action applies only to Table instances. It opens the Browse Data screen for the Table instance; see Viewing Data within the Oracle Life Sciences Data Hub.

Install

This action installs a single object at a time. For Table instances, it performs an upgrade installation.

Launch

This action applies only to Program and Business Area instances. It opens the integrated development environment for the appropriate technology for the object.

Submit

This action applies only to executable instances. The system displays the Submission screen based on the default Execution Setup. You can modify Parameter values as required and submit the job.

Using the Drop-Down List

Additional actions are available in the drop-down list.

1. Select an object.
2. Select an item from the **Select Object and:** drop-down list. The following actions are available:

Add Source Table

This action applies only to executable instances. It allows you to create a Table Descriptor from an existing Table instance for the selected executable instance.

Browse Data

This action applies only to Table instances. It opens the Browse Data screen for the Table instance; see Viewing Data within the Oracle Life Sciences Data Hub.

Copy

See Copying Objects for information. You can select and copy multiple objects at the same time.

Clone

See Cloning Objects for information. You can select and clone multiple objects at the same time.

Default Execution Setup

Displays the default Execution Setup for that instance. You can then modify it; see Creating, Modifying, and Submitting Execution Setups.

Execution Setup

Displays the Execution Setup listing screen for the executable instance. You can then select any Execution Setup defined for the executable and modify or submit it; see Creating, Modifying, and Submitting Execution Setups.

Remove

The system immediately removes the object from the user interface so that it is no longer visible. The object is not deleted from the database until the next Work Area installation, at which point the system automatically drops it. You can select and remove multiple objects at the same time.

> **Note:**
>
> If you remove a Table instance, you lose all the data it contains.

To protect data, when you try to remove a Table instance:

- The system does not remove the Table instance if its validation status is Production.
- If the Table instance is mapped to a Table Descriptor in one or more Programs, Load Sets, Data Marts, or Business Areas, the system displays a warning listing the executable objects to which it is mapped. It also displays the validation status of the Table instance, and whether or not the Table instance is installed.
- If the Table instance is not mapped to any Table Descriptors, the system displays a confirmation message.

> **Note:**
>
> You cannot remove an object that is currently checked out by someone else unless you have the Checkin Administrator role.

View All Outputs

This action applies only to executable objects. It displays a list of all outputs produced by the instance. You can click the links to view the actual outputs.

View Output

This action applies only to executable objects. It displays the most recent output produced by the default Execution Setup for the instance.

Adding Object Instances to a Work Area

To add an object to a Work Area, do the following:

1. From the **Add** drop-down list, select the type of object you want to add.
2. Click **Go**. The system displays the Create screen for the type of object you want to add.

   For instructions on creating objects, see the chapter on each object or click Help from the Create or object properties screen.

Managing Table Instance Snapshot Labels in a Work Area

You can add, remove, or move labels to or from the data in any or all of the Table instances in a Work Area at the same time.

By default, the system displays the most recent snapshots of all audited Table instances in the Work Area that contain data. If you want to work with an older snapshot, perform a query; see Querying for Snapshots.

This section contains the following topics:

- Querying for Snapshots
- Table Instance Information Displayed
- Adding, Removing, or Moving a Snapshot Label

Querying for Snapshots

To add, remove, or move a label from one or more snapshots that are not the most current, start by searching for the Table instance snapshots you want to act on:

1. In the Work Area Properties screen, select **Manage Snapshot Labels** from the **Actions** drop-down list. The system opens the Manage Snapshot Labels screen.
2. **Blinding Status**. Tables that contain blinded data have two partitions, one that contains the real, sensitive, blinded data, and one that contains dummy data. You can apply different labels, one at a time, to snapshots of the blinded and dummy partitions. You must specify whether you want the system to query for blinded or dummy.

   The system always queries for snapshots of nonblinded Table instances (Table instances that do not contain blinded data).
3. Select one of the following:
   - **Most Recent Refresh Timestamp as of**. To search by timestamp, click the button next to Most Recent Refresh Timestamp as of and then select a date and time from the drop-down list.
   - **Snapshot Label**. To search by label, click the button next to Snapshot Label and then select a label from the drop-down list.

4. Click **Search**. The system populates the lower portion of the screen with the Table instance snapshots in the Work Area that satisfy the search criteria.

Table Instance Information Displayed

The system displays the following information for each snapshot that satisfies the search criteria you specified in the upper portion of the screen:

- **Table**. The name of the Table instance of which this is a snapshot.
- **Blinding Status**.
  - If **Blinded**, the action you are about to take affects the blinded partition of a Table instance that contains blinded or unblinded data.
  - If **Dummy**, the action you are about to take affects the dummy partition of a Table instance that contains blinded or unblinded data.
  - If **Not Applicable**, the Table instance does not contain blinded or unblinded data.
- **Writing Instance Type**. The object type that writes data to this Table instance: either a Load Set instance or a Program instance.
- **Writing Instance Name**. The name of the Writing Instance.
- **Job ID**. The unique ID of the job that wrote the data to the Table instance at the time in the Refresh TS column.
- **Snapshot Label**. The snapshot label or labels currently applied to the Table instance snapshot.
- **Source Currency Information**. Currency (timestamp) of the Table instances or external tables or views that the Writing Instance (Program or Load Set) read from when writing to the Table instance snapshot. If the Writing Instance is a Load Set, source timestamp comes from the Load Set execution time for all Load Set types except Oracle Clinical Load Sets; Oracle Clinical type Load Sets get source timestamp from the Oracle Clinical system.
- **Refresh TS**. The timestamp that defines the snapshot of the Table instance.

Adding, Removing, or Moving a Snapshot Label

To add, remove, or move a snapshot label, do the following:

1. The most current snapshots available are displayed by default. If you want to work on older snapshots, enter a query; see Querying for Snapshots.
2. Specify the label to want to add, remove, or move. Either enter it in the Snapshot Label field or, if the label you want to add or remove has already been used in this Work Area, do the following:
   - Click the Search icon for the Snapshot Label field. The system opens a Search pop-up window.
   - Enter the label you are looking for, or use special characters if you do not know the exact label. Special characters are explained in the "Searching" chapter of the *Oracle Life Sciences Data Hub User's Guide*.
   - Click **Go**. The system returns all the labels that fit the criteria.
   - Click the **Quick Select** icon for the label you want.

- The system returns to the Manage Snapshot Label screen with the label you selected in the Snapshot Label field.

3. Click the **Select** checkbox to select the snapshots to which you want to add, or from which you want to remove, the label.
4. Click one of the following:
   - **Add Snapshot Label**. The system adds the label you specify to the snapshot(s) you specify.
   - **Remove Snapshot Label**. The system removes the label from the snapshot(s) you specify.
   - **Add/Move Snapshot Label**. For each Table instance you specify, if the label you specify is currently applied to any snapshot, the system removes the label from that snapshot and moves it to the snapshot you specify.

Personalizing Your Work Area Properties Screen

So much information is available on the objects in the Work Area that it does not fit easily onto a standard computer screen. You can free up space by removing columns you don't use or changing the order in which columns are displayed so that the ones you need most often are easily visible. You can scroll to any columns you do not remove.

You can select an alternative view from the **View** drop-down if other views are available, or create your own custom view. After you create your view, other people can use it too.

To personalize your view of the Work Areas Properties screen:

1. Click the **Customize** button.
2. Do one of the following:
   - If you want to use an existing view without changes, select **Yes** in the Display View column in the view's row. Skip to the last step: click **Apply**.
   - If you want to create a new view beginning with the default values, click **Create View**. The system opens the Update View screen displaying the default settings.
   - If you want to create a new view beginning with an existing view, select a view and click **Duplicate**. The system opens the Update View screen displaying the settings of the view you duplicated.
   - To modify an existing view, click the pencil icon in its Update column. The system opens the Update View screen displaying the settings of the view.
3. Enter values in the following fields:
   - **View Name**. Enter a name for your view or change the existing name.
   - **Number of Rows to Display**: From the drop-down, select a number of rows to display in the Work Area screen at a time. The options are **5**, **10**, **25**, and **50**.
   - **Set as Default**. Check this box to display this view each time you enter the Work Area Properties screen.

- **Description**. (Optional) Enter a description. For example, you can provide information that will help other people decide if they want to base their own view on this one.

4. Remove columns from display (optional). If you want to remove columns, move them from the **Columns Displayed** list to the **Available Columns** list, either by double-clicking on them or by selecting them and using the **Remove** button. You can also use **Remove All** and then add the columns you want using the **Move** button.

> **Note:**
>
> You cannot remove the **Name** column.

5. Change column display order (optional). The left-to-right display order of columns on the Work Area Properties screen is determined by the top-to-bottom order of columns in the **Columns Displayed** list. The topmost column here is displayed farthest to the left. Select a column whose order you want to change and use the up and down arrows to move it relative to the other columns.
6. Change the Sort settings (optional). Use the Sort settings to determine the order in which object instances are displayed by default. You can also change the Sort order directly in the Work Area Properties screen at any time (by clicking on column headings) without affecting the view.
   - **Column Name**. From the drop-down list in the First Sort row, select the column you want to sort on.
   - **Sort Order**. From the drop-down list, select the order you want to use: Ascending or Descending.

   You can define a Second Sort to control the order of objects within the limits of the First Sort, and a Third Sort to control the order of objects within the limits of the Second Sort.
7. **Rename Columns and Set Totalling**. Click the Rename Columns/Totalling button.
   - **Rename Columns**. You can change the displayed heading for most columns. Enter the replacement text in the New Column Name text box next to the default column heading.
   - **Display Totals**. This option is displayed by default for numeric columns. However, it is not useful in these cases to check this option.
8. Click **Apply**.

Installing a Work Area and Its Objects

This section contains the following topics:

- About Work Area Installation
- Running a Work Area Installation
- Installing Individual Objects
- Viewing Installation Results

- What Happens During a Work Area Installation

About Work Area Installation

To use a Table, Program, Load Set, Data Mart, Report Set, Workflow, or Business Area definition that you have created in Oracle LSH, you must create an instance of it in a Work Area and **install** the instance and the Work Area itself. The first time you install a particular Work Area, the system does the following:

- Creates an Oracle LSH Schema—a set of database schemas; see Schemas
- Instantiates Table instances as database tables
- Instantiates Table Descriptors in executable objects and Business Areas as views onto the tables to which they are mapped
- Compiles source code for PL/SQL -type Programs, including PL/SQL Programs contained in Report Sets and Workflows

> **Note:**
>
> SAS and Oracle Reports Programs are compiled at runtime by SAS and Oracle Reports, respectively.

- Associates an Oracle Warehouse Builder Task with every executable object, for use in executing the object

After you install a Work Area, the system keeps it under version control. As soon as you change anything in an installed Work Area—check out an object instance, add or delete an object instance, or even change the description—the system creates a new version of the Work Area that contains all the changes you make before re-installing the Work Area.

> **Note:**
>
> If you install a single executable object using the Install icon or drop-down item the system tries to map any unmapped Table Descriptors and may create target Table instances; see Installing Individual Objects. This functionality is not available when you install at the Work Area level.

Installation Modes

Oracle LSH supports three installation modes: Full, Upgrade, and Partial. See the section on the Omitted flag for further details on how each installation mode handles omitted objects.

- **Full** installation drops and replaces the entire set of schemas. All objects—including tables, checked-out objects, and any objects whose Omitted flag is checked but have been previously installed—are dropped, replaced, and checked in. Full installation deletes all data. During a full installation the entire Work Area is locked and no one can modify any object in the Work Area, including objects whose Omitted flag is checked.

Full installation may be useful during development and quality control testing when data may be corrupted and no audit trail is required. When you change an installed Table's blinding flag, you must do a Full Work Area installation to apply the new blinding status to the Table instance.

- **Upgrade** installation compares the Work Area's installed objects to the current objects and adds new objects and replaces objects that have changed—except for Table instances, which are always upgraded rather than replaced so as to save all data.

  Upgrade installation never drops an object. If you check the Omitted flag for an object and run Upgrade installation, the object is not installed.

  During an upgrade installation the entire Work Area is locked and no one can modify any object in the Work Area, including objects that are omitted from the installation.

- **Partial** installation allows you to specify the action you want the installation job to take on the objects you include in the installation, including explicitly dropping objects. For each table you can specify whether to replace or upgrade the table; if you choose to replace it, the system deletes its data; if you choose to upgrade it, the system upgrades the table to the new version without deleting its data.

  During a partial installation, you and other users can continue to work on the objects that are omitted from the installation.

  Partial installation is useful during development, when multiple developers are working in the same Work Area and need to install and test their objects at different times.

> **Note:**
>
> To safeguard data, Oracle LSH allows only nondestructive installation modes in Work Areas with a Usage Intent of Production. You can use Upgrade mode or Partial mode, but in Partial mode Table instances must have Upgrade as their assigned action.

Installation Rules

The system cannot install a Work Area in the following circumstances:

- The Work Area has a status of Retired.
- The Work Area version is not the most current version.
- The Work Area has a usage intent of Production and the installation mode is set to Full or to Partial with an action other than Upgrade for one or more Table instances.
- The validation status of any object included in the installation is less than the usage intent of the Work Area.
- No objects have changed in the Work Area since the last successful installation.
- A previous attempt to install the same version of the Work Area was unsuccessful, and has not been cancelled by the user.

- The source definition of one or more object instances included in the installation have been explicitly checked out by a user other than the person initiating the installation.

> **Note:**
>
> People with Checkin Administrator privileges can install objects checked out by other people.

- The system cannot install a Program instance or other executable if the Table instances to which it is mapped are not either already installed or included in the same installation.
- If the source code of a PL/SQL Program included in the installation does not compile, the installation fails.

Running a Work Area Installation

To install a Work Area and one or more of the objects it contains, go to the Work Area and do the following:

1. Click **Installation** in the Properties screen of the Work Area you want to install. The system displays the Work Area Installation screen.
2. Choose a mode of installation. See Installation Modes for further information.
   - **Full** installation drops and replaces the entire schema with all objects, including tables, deleting all data.
   - **Upgrade** installation automatically add and replaces only objects that are new or changed since the last successful installation, and always upgrades tables, rather than deleting them, so that it does not delete data.
   - **Partial** installation allows you to specify which objects you want to install and what action you want the installation job to take on the objects you include.

> **Note:**
>
> To safeguard data, Oracle LSH allows only nondestructive installation modes in Work Areas with a Usage Intent of Production. You can use Upgrade mode or Partial mode, but in Partial mode Table instances must have Upgrade as their assigned action.

3. In the **Install Actions** drop-down list, select **Process Current Installation to Completion**. This is the only option currently available.

   Choose whether or not to view the installation scripts before committing the changes to the database. The scripts list the object instances to be installed with the DDL scripts detailing the actions to be taken...There's a list of Install Actions in the FDD
4. Choose whether or not to run a batch install:

- If you select **Batch Install**, the system runs the installation as a batch process. The installation job is placed on a queue and the system returns control of the UI to the user. You can abort the installation if necessary.
- If you do not select **Batch Install**, the system runs the installation in interactive mode. The installation job does not go onto a queue and therefore can run more quickly than in batch mode. However, you cannot do anything else in Oracle LSH until the installation has completed. You cannot abort the installation. Interactive mode is appropriate when you are installing a small number of objects.

5. Choose whether or not to **Force Script Regeneration**. During installation, the system generates two sets of scripts for objects that have changed since the last successful installation:

   - DDL scripts for each object included in the installation. The system uses these scripts to create the actual schema objects during installation.
   - Scripts to be used at runtime for some of the objects; for example, a SAS script for a SAS-based Program that establishes the appropriate SAS views for accessing Oracle LSH data.

   If you select this option, the system generates new scripts for all objects in the installation, even if they have not changed. All objects are either replaced or, if they are Tables, upgraded. This is useful if your schema has become corrupted and you want to recreate all objects without losing any data.

   Checking this attribute has an effect only in Upgrade mode. In Full mode the system always regenerates scripts for every object regardless of the setting of this attribute. In Partial mode the system uses the action you specify in the **Actions** column for each object to determine whether to regenerate the scripts.

6. Review the objects to be installed. You can sort the objects in the Work Area by clicking most of the column headings, including: **Name**, **Type**, **Installable**, **Upgradable**, **Definition Checked Out By**, and **Current Version Installed?**.

7. Check the box in the **Omitted** column for each object you want to omit from the installation. The system automatically checks this box for objects whose status is Non Installable. You can also use the **Omit All** and **Omit None** buttons.

   You can omit any object in any installation mode and, if the object you omit has never been installed, it will be truly excluded from the installation process. However, if it has already been installed, the system takes a different action depending on the installation mode:

   - **Full**. In Full mode, the installation process drops the object and replaces it with the same version that was already installed. **If a Table instance has already been installed** and you run a Full installation on its Work Area, the system drops and replaces the Table instance and **deletes all its data**, even if you check its **Omitted** checkbox.
   - **Upgrade**. In Upgrade mode, objects marked as **Omitted** are not installed.
   - **Partial**. In Partial mode, when you check an object's **Omitted** checkbox, the system automatically sets the **Action** for the object to No Action. If you leave Action set to No Action, the object is omitted. If you change this setting, the installation does not omit the object, but performs the action you specify.

> **Note:**
>
> Objects with a value of No in the Installable column are automatically omitted from the installation in all modes.

8. To save your changes but install later, click **Apply**.

   To install now, click **Apply and Install**.

> **Note:**
>
> Before the installation can begin, the system must wait for all job executions of Work Area Programs, Load Sets, Report Sets, or Workflows currently running to complete. The system prevents new jobs from starting.

Installing Individual Objects

You can install any object by clicking the Install icon in its row. If the object is not installable the system automatically performs automatic mapping by name for any unmapped target Table Descriptors, looking for Table instances of the same name in the current Work Area. If any target Table Descriptors remain unmapped, the system creates Table instances from them in the current Work Area and maps them. The executable object may then be installable; see Installation Requirements for Each Object Type.

If the object is installable, the system performs an upgrade installation for the selected instance and any of the Table instances to which it is mapped that are not already installed. If the object being installed or any of its mapped Table instances is not installable, the installation fails and the system displays an error message with the name of the noninstallable object.

If the object being installed is an Oracle Clinical SAS or Oracle Data Extract Load Set and there are no target Table Descriptors defined, the system automatically creates target Table Descriptors, Table instances, and Table definitions based on all the active SAS or Oracle Data Extract views defined for the selected Oracle Clinical study or study set, and maps the Table Descriptors and corresponding Table instances.

Viewing Installation Results

When you start an installation, the system displays the Installation Processing Monitoring screen. It has the same information as the Installation screen. If you are running installation in interactive (not batch) mode, all the fields are read-only. In batch mode you can use the **Abort** button to stop the installation job.

When the installation completes, a different display appears depending on whether the installation completed successfully or failed; see:

- Failed Installations
- Successful Installations

Failed Installations

If the installation fails, the system displays the status Install Failed. The Installation Status displays the last phase the installation process reached. See Installation Rules for some of the reasons an installation may fail, and Work Area Installation Phases and Statuses.

> **Note:**
>
> You must cancel the installation by clicking **Cancel** or **Cancel Installation** before you can try again to install the Work Area.

You cannot roll back any changes made by the installation process.

To see what actions the installation took, look at the log file.

Finding the Log File

To view the log file, do the following:

1. Click **Cancel** to return to the main Work Area screen.
2. From the **Actions** drop-down list, select **Installation History**.
3. Click **Go**. The system displays the Installation History screen.
4. Click the **View Log** link for the most recent installation attempt. The system displays the log file.

Reading the Log File

In the log file, you see a record of all the phases the installation process passed through before failing. See Work Area Installation Phases and Statuses for information on these phases.

The log file contains information about parts of the installation that failed, with an error message. Each failed activity is called a "unit." This is an Oracle Warehouse Builder term and can be a package or other unit.

For example, if the installation fails because the PL/SQL code in a Program did not compile, the log file contains a message like the following:

```
Calling Create deployment
Unit Name=PKG_CDR_W4_170DAE141_1_$CREATE_7
ORA-00900: invalid SQL statement
Unit Name=PKG_CDR_W4_170DAE141_1_$CREATE_7
ORA-04042: procedure, function, package, or package body does not exist
fail to complete create deployment
Set install status to $INSTSTATUSES$CREATEACTIVE
set work area status to install failed
```

Note the name of the unit and then look at the bottom of the log file for further details about the problem(s) with the unit.

Successful Installations

If the installation succeeds, the system displays a status of Installed and an installation status of Installation Completed Successfully.

Each object included in the installation is displayed, and you can click the **View Script** link for any object to see the DDL scripts the system generated for that object.

Click **View Installation Log** to see the log file.

In Oracle LSH Release 2.5 the **Abort** and **Resume** buttons are inactive.

What Happens During a Work Area Installation

During installation, Oracle LSH uses Oracle Warehouse Builder (OWB) to convert Work Area definitional objects to database objects.

This section contains the following topics:

- Schemas
- Object Name Resolution
- Work Area Objects Converted to Oracle LSH Schema and OWB Objects
- Work Area Installation Phases and Statuses
- Work Area Status

Schemas

Each Work Area is installed to its own set of Oracle database schemas, collectively called an Oracle LSH schema. An Oracle LSH schema encompasses all the installation targets, including Oracle database schemas and file systems.

and Discoverer Plus EULs (created for Work Areas that contain Business Areas for visualizations).

During installation the system determines what schemas are required and creates them.

Each Oracle LSH schema includes a primary Oracle database schema and one or more auxiliary Oracle database schemas. Auxiliary schemas contain compiled PL/SQL code and private synonyms to resolve naming conflicts.

Primary Schema

Each Oracle LSH schema includes one primary schema that owns all installed database objects except the compiled PL/SQL code. The primary schema is name W*cccc_nnnn*, where *cccc* is the company ID and *nnnn* is the internal hexadecimal ID of the Work Area.

Auxiliary Schemas

Auxiliary schemas contain all the Work Area's compiled PL/SQL code and private synonyms that resolve the object instances referenced in the PL/SQL code to the actual Tables in the primary schema or in the primary schemas of other Work Areas (seeObject Name Resolution).

The auxiliary schemas are named W*cccc_nnnn_x*, where *nnnn* is the internal hexadecimal ID of the Work Area and *x* is a unique identifier for the auxiliary schema.

Business Areas have a similar naming scheme but have _d appended

Object Name Resolution

Because you can reuse Oracle LSH object definitions such as Tables, Table Descriptors, and Programs in Oracle LSH, there can be naming conflicts within a Work Area. For example, two PL/SQL Programs may refer to two different Tables called Adverse Events, one of which contains raw source data, while the other contains transformed data.

Oracle LSH resolves any naming conflicts in PL/SQL programs (including Oracle Reports Programs and including Report Sets, Workflows, Load Sets, and Data Marts) by using private synonyms in as many auxiliary schemas as necessary. For each PL/SQL Program, the system records the name of the auxiliary schema that resolves its Table Descriptors' naming conflicts.

For security reasons, PL/SQL Programs cannot be stored in the primary schema. They are stored in an auxiliary schema. If a PL/SQL Program includes multiple packages, they are all installed in the same auxiliary schema.

The system analyzes the PL/SQL Programs to be created for a Work Area, determines the number of auxiliary database schemas needed, and creates them. This analysis is performed in Oracle LSH metadata tables during the installation Preparation in Progress phase; the actual auxiliary schemas and their private synonyms are created during installation.

Work Area Objects Converted to Oracle LSH Schema and OWB Objects

Oracle Warehouse Builder converts defined objects in a Work Area to Oracle LSH schema objects as follows:

Table Instance

A Table instance becomes a database table, with any user-defined constraints and any system-defined or user-defined indexes as well as a data manipulation view, an associated Instead-Of trigger, and supporting PL/SQL package.

If a Table instance is defined as a pass-through view, it becomes a view in the database.

Executables

For each executable object—Program, Load Set, Report Set, Workflow, or Data Mart—the system creates its Table Descriptors as views onto the Table instances to which they are mapped and generates a PL/SQL package.

Source Table Descriptors each become a view that does the following:

- resolves the Table Descriptor to the real table
- maps Table Descriptor Columns to real table columns
- implements Oracle LSH's dynamic security mechanism
- implements currency snapshotting
- implements incremental data access

- implements blinding

Target Table Descriptors each become a view that does the following:

- resolves the Table Descriptors to the DML View
- maps Table Descriptor Columns to underlying table columns
- implements Oracle LSH's dynamic security mechanism

PL/SQL Programs, including those contained in Report Sets and Workflows, become:

- compiled PL/SQL code in an auxiliary schema or schemas
- synonyms to map names used in the PL/SQL Program to the proper source or target view in the primary schema
- synonyms to map names of external package references to packages in another auxiliary schema

For SAS and Oracle Reports Programs, OWB creates an OWB Task to call the DP Server and to grant access to the appropriate database tables.

Work Area Installation Phases and Statuses

The Work Area installation process includes many phases. As the job progresses through these phases the system assigns it different installation statuses. The system displays the highest status reached during the process when the process is complete. You can also see the job's progress from phase to phase in the log file.

After completing each phase, the system checks the value of the Abort flag. If the installation is running in batch mode and the user has clicked the Abort button to stop the installation, the system stops the job. You cannot roll back any changes the job may have made up to that point.

Installation Specification

The system gathers the attribute settings for the installation: Batch Mode (Yes/No), Force Script Regeneration (Yes/No), Installation Mode (Full, Upgrade, Partial), Install Status of the previous installation, if any (statuses noted after each phase in this list), Installation Number (current installation number for this Work Area version plus one).

Related Installation Status: SPECIFIED (Specification phase complete)

Preparation

The system performs all tasks required before the Generation Phase, including obtaining the necessary locks, identifying the objects to be installed and the actions to be taken on them, checking in objects, and ensuring that the installation is valid (see Installation Rules).

user will be able to request an Impact Report -- created during Prep

Related Installation Status: PREPARED (Preparation phase complete)

Generation

In a full or upgrade installation the system determines how many schemas are necessary to hold the installed objects and which objects to place in which schema. The system generates the DDL scripts that will be used to install, drop, drop and

replace, or upgrade each object included in the installation and, in a full or upgrade installation, the schemas themselves.

Related Installation Statuses: GEN_ACTIVE (Generation phase in progress), GENERATED (Generation phase complete)

Quiescence

If the Work Area is currently being installed, the system prevents jobs from starting that are based on the submission of Oracle LSH Programs, Load Sets, Report Sets, Workflows, or Data Marts installed in the Work Area, and waits for any jobs currently running to complete. In a partial installation, the system prevents or waits for job executions only of the objects being installed; the execution of objects not included in the partial installation can proceed.

Related Installation Status: QUIESCING (quiescing runtime processing)

Unit Definition

Oracle LSH calls the Oracle Warehouse Builder (OWB) interface packages to create the deployment units that OWB will use to carry out the required actions on each object in the installation.

Related Installation Statuses: DEF_ACTIVE (Unit Definition phase in progress), DEFINED (Unit Definition phase complete)

Schema Activities

The system creates the Oracle schemas required by the installation if they do not yet exist. In a full installation the system drops existing schemas creates new schemas.

Related Installation Statuses: SCHEMA_ACTIVE (Schema Activities in progress), SCHEMA_COMPLETE (Schema Activities complete)

Upgrade Prepare

OWB compares the proposed table and table-related changes to the state of the objects in the database schema and prepares an upgrade plan script to implement the upgrade. This phase applies only to upgrade installations and partial installations where the action on an object is Upgrade.

Related Installation Statuses: UPGPREP_ACTIVE (Upgrade prepare in progress), UPGPREPARED (Upgrade prepare complete), UPGPREP_FAILED (Upgrade prepare failed)

Upgrade Deploy

OWB carries out the upgrade plan and stores the initial state of the tables and their data in backup tables. This phase applies only to upgrade installations and partial installations where the action on an object is Upgrade.

Related Installation Statuses: UPGDEPL_ACTIVE (Upgrade deployment in progress), UPGDEPLOYED (Upgrade deployment complete), UPGDEPL_FAILED (Upgrade deployment failed)

Upgrade Finalization

OWB discards the backup tables. The upgraded tables contain the data. This phase applies only to upgrade installations and partial installations where the action on an object is Upgrade.

Related Installation Statuses: UPGFIN_ACTIVE (Upgrade finalization in progress), UPGFINALIZED (Upgrade finalization complete)

Drop

OWB performs the deployment units for any dropped objects.

Related Installation Statuses: DROP_ACTIVE (Drop phase in progress), DROPPED (Drop phase complete)

Create

OWB performs the deployment units for any objects being created.

Related Installation Statuses: CREATE_ACTIVE (Create phase in progress), CREATED (Create phase complete)

Completion

The system performs the following clean-up tasks:

- For full and upgrade installations the Work Area Status is set to Installed. If there are any objects that are checked out in the Work Area (due to being omitted from the installation) the system then checks out the Work Area and changes its status to In Definition.
- For partial installations, if the Work Area contains omitted objects whose current version has not been installed, the Work Area is checked in with a status of Partial Install and checked out again with a status of In Definition.
- If the Parameters or Parameter settings for a newly installed program have changed, existing repeating, deferred, or backchain submissions are cancelled without notification.
- The system releases the state of quiescence.
- The system releases the installation lock.

Related Installation Status: INSTALLED (Installation completed successfully)

Work Area Status

As the Work Area moves through its life cycle of development, installation, modification, reinstallation, and retirement, the system assigns it a Work Area Status as follows:

In Definition

The Work Area has been defined but not yet installed, or has been installed and is now being modified.

Locked for Installation

The Work Area and all its object instances are locked. No one can check them out or run any jobs. This status applies to full and upgrade installations.

Locked Partial

The object instances included in a partial installation are locked. No one can check out locked objects or submit locked executables for execution.

Partial Install

The Work Area has never had a full installation but has been through a successful partial installation.

Installed

The Work Area has been successfully installed and is available for use.

Retired

The Work Area still exists in the database but you cannot submit any of its executable objects for execution; see Retired for further information.

Cloning Work Areas for Testing and Production

Oracle LSH supports cloning Work Areas so that you can make an exact copy of a Work Area and all the objects it contains and install it to a new Oracle LSH Schema to create a test or production data environment.

When you first create an object of any kind, including a Work Area, its validation status is set to Development. Your company should develop standards for promoting objects to the validation status Quality Control, and from Quality Control to Production. Oracle LSH enforces rules concerning the interaction of the validation statuses of a Work Area and each of its objects, and the value of the Work Area's Usage Intent attribute, to support the validation process.

Tools

Oracle LSH uses the following tools to support validation and separate environments for testing and production:

- **Usage Intent Attribute**. Work Areas have attribute called Usage Intent with the possible values Development, Quality Control, and Production.
- **Cloning**. Work Areas have an operation called cloning that creates an exact duplicate of the Work Area and all its object instances.
- **Validation Status**. All object definitions and instances, including Work Areas, have a validation status with the possible values Development, Quality Control, Production, and Retired.

When all the object instances in a Work Area, and the object definitions on which they are based, reach a higher validation status than the Work Area's Usage Intent, a user with the necessary privileges can clone the Work Area and set the new Work Area's Usage Intent attribute to the next higher value. See Figure 12-1.

Rules

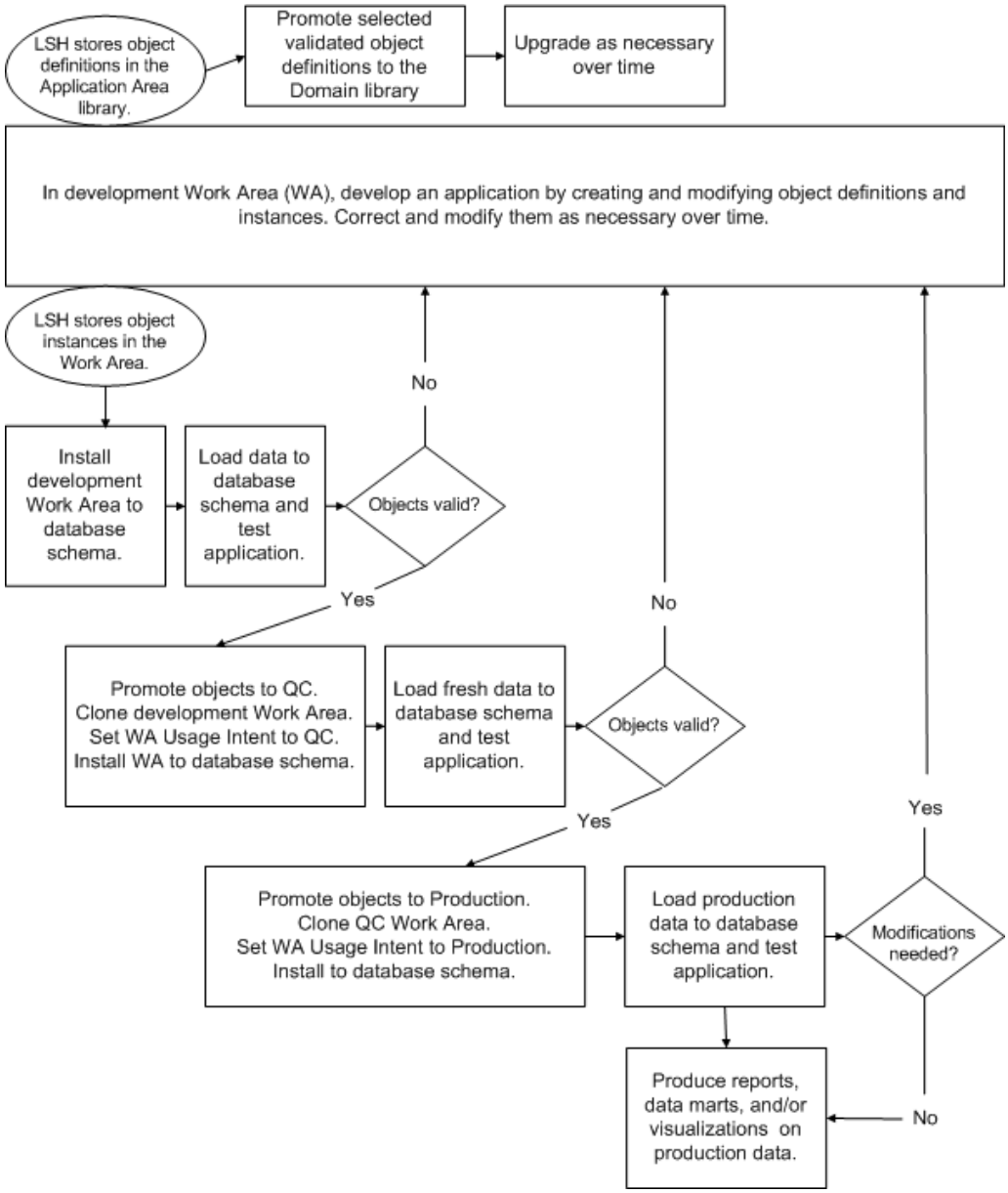
Oracle LSH enforces the following rules:

- To be installed in a Work Area, object instances must have a validation status equal to or greater than the Usage Intent of the Work Area.
- A Work Area cannot be promoted to a validation status higher than the validation status of any of its object instances.
- No executables can be run in a Work Area until the Work Area's validation status is equal to or greater than its Usage Intent value, except by users with the special Install Qualify Submit privilege on the Work Area. This is to allow testing of a Work Area before making it available to the full set of users with security access to it.
- Full installation is not allowed in Work Areas with a Usage Intent of Production. This is to protect production data from deletion.
- Application Life Cycle
- Cloning a Work Area

Application Life Cycle

The intended Work Area usage includes the following stages: Development, Quality Control, and Production. Figure 12-1 shows these stages.

Figure 12-1 Application Development and Validation Process



Development

When you create a Work Area, the system sets both its Usage Intent attribute and its validation status to Development. When you create an object definition or instance, the system sets its validation status to Development. When you have successfully conducted unit testing on an object, according to your organization's standards, set the object's validation status to Quality Control (or request a privileged user to change the status, depending on your security design).

When all the object instances in a Work Area have a validation status of Quality Control, a privileged user clones the Work Area, creating duplicates of all object instances, with pointers to the same object definitions and the same validation statuses as the originals. The version number of the new object instances is 1. The

system creates a label for both Work Areas indicating that they are identical at the time of cloning. The privileged user enters text for the label and creates the new Work Area with a Usage Intent of Quality Control. However, its validation status should remain at Development.

Quality Control

Install the new Quality Control Work Area. While its validation status (Development) is lower than its Usage Intent (Quality Control), only a user with the Install Qualify Submit privilege can run executables. That privileged user loads fresh data and tests the installation. The privileged user then promotes the Work Area validation status to Quality Control. Users with normal security access to the Quality Control Work Area can then test the objects.

If testers find bugs or other problems, developers should fix them in the Development Work Area. To do this, you can clone the QC Work Area onto the development Work Area, overwriting the old one, perform a full installation, and load fresh data. When all objects have been fixed and tested, and their validation status upgraded, you can clone the development Work Area onto the QC Work Area.

> **Note:**
>
> If new objects are being developed in the Development Work Area, do not clone the QC Work Area onto the original Development Work Area or they will be lost. Instead, do one of the following:
>
> - Create a new Work Area and clone the QC Work Area onto the new Work Area. Give the new Work Area a unique name such as "Post-QC Development." Copy and paste new objects that are ready for testing into the Post-QC Development Work Area before cloning it onto the QC Work Area for testing.
> - Clone the current Development Work Area onto the QC Work Area. Test only objects whose validation status is set to QC.

Promote each object definition and instance to Production when it meets your production standards. Clone the QC Work Area to create a Production Work Area.

Production

Install the new Production Work Area. While its validation status is lower than its Usage Intent, only a user with the Install Qualify Submit privilege can run executables. That privileged user loads fresh data and tests the installation. The privileged user then promotes the Work Area validation status to Production. Users with normal security access to the Production Work Area can then run the application.

Run the application as necessary for your business purposes. Using the same cloning procedures described above, modify the objects as necessary over time. Make modifications through the Development Work Area, test each modified object definition in the QC Work Area, and clone the QC Work Area to the Production Work Area.

> **Note:**
>
> The cloning operation replaces only objects that have been modified. Cloning does not replace Production Table instances or data.

Retired

When you close a trial, you can set its Production Work Area's validation status to Retired. This prevents anyone from running any executables within the Work Area. However, you can still use the data as input to a Program or other executable in a different Work Area. For example, you can merge and analyze the data with data from other closed or current trials.

When you retire a Work Area, you may also want to change its user group assignments so that only a very limited group of people can change its validation status back to Production and run Programs or other executables on the data.

If you need to run a Program in a Production Work Area after you have set its validation status to Retired, you must set its validation status back to Production. For example, to track patients and update their adverse event records after the trial has closed, set the Work Area's validation status back to Production, run the necessary Programs, and set the validation status back to Retired. Alternatively, create a new Work Area for this purpose.

Cloning a Work Area

You can use the cloning operation together with the Usage Intent Work Area attribute and object validation statuses (Development, Quality Control, Production, and Retired) to create clean, controlled, distinct environments for application development, testing, and production. See Application Life Cycle.

The cloning operation is similar to the copy operation except that it is possible to clone a Work Area over an existing Work Area, so that the clone overwrites the existing Work Area. For example, if you already have a Quality Control usage intent Work Area, and several objects fail quality control testing, you can update the objects in the Development Work Area and then clone the whole Development Work Area onto the QC Work Area, creating a new version of the QC Work Area.

Object instances in the Work Area clone point to the same source definitions in the same location that the instances in the original (source) Work Area did.

> **Note:**
>
> If you clone onto a Work Area that contains objects, the cloning operation replaces only those objects that have a different version number from the same object in the source Work Area.

When you clone a Work Area, you specify a label that the system applies to the source and target Work Area version. The label is proof that the source and target Work Area versions are identical. All the object instances in the two Work Areas are identical, and

corresponding objects' validation status is the same. If you clone the same version of the same Work Area more than once, the system uses the same label each time.

To clone a Work Area, do the following:

1. You can begin a cloning operation in two different places:
   - In the original Work Area's Properties screen, click **Clone**.
   - In the main Application Development screen, click the icon in the **Clone** column for the original Work Area.

   The system opens the Step 1 Clone Work Area screen.
2. In the **Clone Label** field, enter the text for the label. This text will be displayed in the relevant version of the source and target Work Areas.
3. From the Usage Intent drop-down, select the Usage Intent you want to set for the target Work Area.
4. In the **Clone Destination** area, click the button in the **Select** column to specify the target.

   If you select an Application Area, the cloning operation creates a new Work Area in that Application Area. The new Work Area's name is the same as the source Work Area with "_1" appended.

   If you select an existing Work Area, the cloning operation overwrites that Work Area.
5. Click **Review**. The system displays information about the source (original) and target Work Areas, including all object instances in the source, for you to review. For each object, the system displays the action to be taken in the target Work Area:
   - **Create**. If the object does not exist in the target Work Area, the system creates it.
   - **Replace**. If the object exists in the target Work Area and has been modified in the source Work Area since the last clone, the system replaces the target object with the source object.
   - **Remove**. If the object exists in the target Work Area but not in the source Work Area, the system removes the object in the target Work Area.
   - **No Action**. If the object exists in the target Work Area and has not been modified in the source since the last clone, the system does not modify the target object.

   If you see a problem, click **Cancel** to cancel the cloning operation and return to the Work Area.
6. To run the cloning operation, click **Finish**. The system performs the cloning operation and opens the Properties screen of the target Work Area, displaying a confirmation message that the clone was successful.

13

Execution and Data Handling

This section contains information on the following topics:

- About Execution
- Submitting Jobs for Execution
- Data Processing Types
- Data Auditing, Snapshots and Refresh Groups
- Processing Data Subsets
- Forward Chaining
- Backchaining
- Managing Blinded Data
- Using Message-Triggered Submission from External Systems

About Execution

The Oracle Life Sciences Data Hub (Oracle LSH) offers the following capabilities related to execution and data handling:

- You can submit a job for immediate execution or schedule it for a later date and time or set it up to run at regular intervals; see Submitting Jobs for Execution.
- You can set up and run backchain execution to ensure that you are processing the most current data available in the source data system; see Backchaining.
- You can set up your source data system to send XML messages to trigger the execution of a job in CDR. NOTHING WRITTEN ON THIS AND NO INFO.
- If you are using Reload processing, you can choose to run in either Full or Incremental mode; use Incremental mode to quickly load and changed data; use Full mode to do the same and also to delete records that are not reloaded; see Reload Processing.
- Apply snapshot labels to the target or both source and target Table instances that a particular job reads from and writes to; see Data Snapshots.
- Maintain blinded data; see Managing Blinded Data.

Each job submitted from the user interface is called a master job. A master job may execute a single executable object, such as a Program, Load Set, or Data Mart. When a user submits a Report Set, the master job includes the execution of all the Program instances contained in the Report Set. A Workflow execution may include the execution of other executable objects. A Program execution using backchaining is also a master job that includes multiple subjobs.

Oracle LSH uses the runtime platform of the Oracle Warehouse Builder for all internal execution. Some additional information is available in "Stopping and Starting Services and Queues" in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

Submitting Jobs for Execution

You execute each type of Oracle LSH executable object—Programs, Load Sets, Report Sets, Workflows, and Data Marts—in the same way.

Before you can submit any executable object, you must do the following:

- Define an Execution Setup for the executable instance. See Creating, Modifying, and Submitting Execution Setups for instructions.
- Map the executable instance's source and target Table Descriptors to Table instances; see Mapping Table Descriptors to Table Instances .
- Install the instance and all the Table instances to which it is mapped; see Using, Installing, and Cloning Work Areas.
- Ensure that the source Table instances contain data.

After you have submitted a job, you can track its progress and see details about the job in the Job Execution section of your My Home screen. For information about the job information displayed, see "Tracking Job Execution" in the *Oracle Life Sciences Data Hub User's Guide*.

- Submitting a Job from the Applications Tab

Submitting a Job from the Applications Tab

While you are defining an executable object, it's easiest to submit it directly from the instance in its Work Area. You can also submit jobs from your My Home screen and from the Reports tab; see "Generating Reports and Running Other Jobs" in the *Oracle Life Sciences Data Hub User's Guide*.

To submit a job from the Applications tab, do the following:

1. In the appropriate Work Area, navigate to the installed executable instance you want to submit.
2. Click **Run**. The Submission screen opens.

   Alternatively, in the **Actions** drop-down list, click **Execution Setups**, then Click the **Submit** icon of the Execution Setup you want to use. The Submission screen opens.
3. Set values for Submission Details, Submission Parameters, and Data Currency as necessary. For information on each of these, see Creating, Modifying, and Submitting Execution Setups.
4. Click **Submit**. The system submits the job for execution and displays the job ID.

Data Processing Types

Oracle LSH offers four different basic types of data processing to accommodate the different technologies used for loading data into Oracle LSH and operating on data within Oracle LSH: transactional, reload, staging, and transactional high throughput. The types are described in the following sections.

The type of processing the system uses depends on the processing type specified for the target Table instance, not the Program. However, you must be careful to choose a

processing type for a Table instance that is compatible with the Program that writes to it:

- If the processing type is Transactional, the source code must explicitly update, insert, and delete records.
- If the processing type is Reload, the source code must simply ensure that all records from the source are loaded (as Inserts) into the target Table instance.

Program instances need to be compatible only with Table instances they write to. Any Program instance can read from a Table instance of any processing type.

Although only one Program instance can write to a particular Table instance, some data processing types allow more than one execution of that Program to run at the same time.

This section contains the following topics:

- Processing Types Summary
- Transactional Processing
- Reload Processing
- Staging Processing
- Transactional High Throughput Processing
- SQL*Loader Processing for SAS Programs
- Using Tables as Pass-Through Views

Processing Types Summary

Each processing type is described in the following sections. The following table summarizes the differences among Oracle LSH data processing types in terms of whether they require Unique or Primary Keys, are audited, and the kind of data deletion that they effect:

Table 13-1 Summary of Oracle LSH Data Processing Types

Processing Type	UK/PK Required ?	Audited ?	Data Deletion
Transactional with Audit	Yes	Yes	Soft-deletes data specified by the Program with explicit DML Delete statements.
Transactional without Audit	No	No	Hard-deletes data specified by the Program with explicit DML Delete statements.
Reload	Yes	Yes	In Full mode, soft-deletes all data not reinserted. In Incremental mode, does not delete data.
Staging with Audit	No	Yes	Deletes all data immediately before the next Program execution; saves a copy of the data.
Staging without Audit	No	No	Hard-deletes all data immediately before the next Program execution.
Transactional High Throughput	No	No	Truncates the Table in Full mode. Hard-deletes data with explicit DML Delete statements in the Incremental mode.

- Reload and Transactional with Audit types require either a Primary Key or a Unique Key to be defined for the Table instance. Staging and Transactional without Audit types do not require a Primary or Unique Key.
- Transactional, Reload, and Transactional High Throughput Table instances must be mapped to target Table Descriptors of the same type. Staging Table instances can be mapped to target Table Descriptors of any processing type.

  From the point of view of the Program writing to the Table instance, a Transactional target Table Descriptor can be mapped to either a Transactional or Staging Table instance and a Reload target Table Descriptor can be mapped to either a Reload or Staging Table instance. Staging Table Descriptors must be mapped to Staging Table instances.
- All types except Staging are serialized, so that only one job can write to the Table instance at a time. For staging Table instances, if more than one job writes to the Table instance at a time, the system ensures that there is at least one second's difference in the refresh timestamp.

Different processing types handle data deletion differently:

- **Transactional processing** deletes only those records specified in Delete statements in the Program source code. If the Table instance is audited, the system only soft-deletes the data.
- **Incremental Reload processing** never deletes data. **Full Reload** processing soft-deletes all records not explicitly reloaded.
- **Staging Without Audit processing** deletes all the data written by the previous job at the beginning of the next job. **Staging With Audit** does not delete data.
- **Transactional High Throughput processing** truncates the Table in the Full mode. In the Incremental mode, hard-deletes data if the Program source code specifically issues a Delete statement.

Transactional Processing

In transactional processing, the Program writing to the Table instance does the work of determining which records are inserted, updated, and deleted. The system populates the target database table using explicit DML statements (Insert, Update, or Delete) in the Program's source code. The system processes only those records that the Program specifies; records not explicitly inserted, updated, or deleted are not processed. If a record is explicitly updated but if the data remains the same, the system updates its refresh timestamp.

The Program writing to the table can "see" all current data, including changes it has made and records from previous jobs that are still current. Only one job can write to a Transactional table at a time. The system serializes them.

There are several data processing types that use transactional processing:

- **Transactional with Audit**. The system maintains a record of all changes to all records, including deleted records, over time. Only one Program instance at a time can write to a Table instance of this type.
- **Transactional without Audit**. The system maintains only the current set of records. Only one Program execution at a time can write to a Table instance of this type.

Reload Processing

In Reload processing, the internal data processing algorithm does the work of determining which records to insert, update, and delete. The Program writing to the Table instance simply inserts data (and must insert all data from the source). The system compares the unique or primary key of each inserted record to the existing records to determine whether the record insertion is treated as an insert or an update: if a record already exists with the same unique or primary key, the system treats the loading of that record as an update. The way the system processes deletions depends on whether the job is run in Full or Incremental mode (see below).

If the updated record does not include any data changes, the system simply changes its refresh timestamp to the timestamp of the current job.

Reload processing requires a primary or unique key defined for the target Table instance. If both exist, you must specify which one to use for data processing. It is not necessary to have a primary or unique key defined on the source; for example, a Load Set can successfully load SAS data into an Oracle LSH Table instance even if no primary or unique key is defined for the dataset in the source SAS system.

Programs writing to reload tables can "see" only data processed in the current job.

Reload Table instances are always audited. Only one job can write to a Reload table at a time. The system serializes them.

There are two modes of reload processing: Incremental and Full. The mode used is set in the Execution Setup for the Program writing to the Reload table. It can be bound or settable at runtime by the person submitting the job.

- **Incremental Reload**. In incremental processing, the system never deletes a record. If a record is not reloaded it remains in the system but its timestamp is not updated.
- **Full Reload**. Full reload processing is the same as incremental processing except for one additional step: after processing all the records, the system soft-deletes all records that were not reloaded. The system keeps soft-deleted records in the database associated with an end timestamp and inserts an additional row to explicitly record the deletion.

If you are incrementally adding records to a table, or loading updated versions of different subsets of data, choose Incremental. If you are reloading a complete set of the most up-to-date records that may be missing some records due to deletions, choose Full

Incremental Reload Example: Use Incremental Reload processing for the target Table instance of a Load Set that loads new lab data every week. Do not use Full Reload for loads that contain only new data, because Full Reload would delete all the data not explicitly loaded.

Full Reload Example: Use Full Reload processing occasionally for cleanup with jobs that normally use Incremental Processing. For example, use Incremental Reload to load the external demography table nightly, but use Full Reload on the same table once a month before running a monthly report, to delete any records that have been deleted from the external table.

Staging Processing

Staging Tables are compatible with any type of Program; that is, you can map a Program target Table Descriptor of any type—Staging, Reload, or Transactional—to a staging Table instance.

Staging processing is designed specifically to hold data only for the duration of the execution of the Program or Load Set that writes to the Table instance. If the Program to which a staging Table instance is mapped is contained in a Report Set of Workflow, the system retains the data for the duration of the master job (the whole Report Set or Workflow). However, you can choose to audit a staging Table instance, so that its data is never hard-deleted:

- **Staging with Audit**. In staging processing with auditing, the system does not delete any data; the records remain in the table with a creation timestamp equal to the refresh timestamp of the job that inserted them. The staging processing logic allows a job to process only records whose creation timestamp is equal to its own timestamp. Records in a staging Table instance do not have an end timestamp.

  There is no relationship between records with one creation timestamp and those with another, whether or not they share a unique or primary key. For example, loading the same set of records twice will result in the table containing two complete sets of the loaded records.
- **Staging without Audit** In staging processing without auditing, immediately before the next execution of the Program, the system hard-deletes the data in the table. This setting saves space in the database.

More than one job can run on a staging table instance at a time. Each job can "see" only the records whose creation timestamp is equal to its own refresh timestamp.

Example: You might want to audit data in a staging table that you use for reports on different subsets of data; for example, a Table that holds Adverse Events data that you report in groups according to patient age: one report for Patients in their 20s, another for those in their 30s, and so on.

Transactional High Throughput Processing

This is an audit-less processing type, specifically designed to load large volumes of raw data as quickly as possible. It also supports DML statements.

> **Note:**
>
> If you convert a Transactional High Throughput data processing type Table into a Table with any of the other processing types (by modifying the Table's Process Type attribute), you are required to perform a full install of the Table when you install the Work Area next. This results in the re-creation of the Table and you lose all the existing data.

This processing type supports full and incremental data loading.

You can load large volumes of data by splitting the data loading job and yet be able to view all of the data together in a single snapshot.

This processing type has the following features:

- **Displays All Data in a Single Snapshot**. This data processing type makes all of the data available in a single snapshot even if the data is loaded through multiple jobs (when you use the Full data loading mode).
- **Supports Blinding**. This data processing type provides full support for data Blinding. You can mark the data loaded into the target Table of this processing type as Blinded or Dummy in the same way as with the other data processing types.

  > **Note:**
  >
  > If the table is blinded and data has been loaded into both partitions then performing a load using full mode will delete the data from both partitions. You will lose all Blinded data if you choose the full mode for loading data, even if you run the job with the Dummy Blind Break setting. The system warns you if a new data-loading job is about to overwrite Blinded data giving you the option to cancel the job.

- **Supports Full and Incremental Data Loading Modes**. This processing type supports both incremental and full data loads. The default mode is full. In the full mode, the system truncates the existing Table and loads fresh data into it. In this mode, you will lose all the Blinded data in the Table, even if you run the job with the Dummy Blind Break setting.

  In the incremental mode, the new data is appended to the Table. The system hard-deletes data only if the Oracle LSH Program explicitly issues a Delete statement.

  > **Note:**
  >
  > The default data loading mode is Full. If you do not want to lose all your existing data, change the data loading mode to Incremental.

- **Supports Compression**. If you create Oracle LSH Table instances marked with Transactional High Throughput data processing type in a tablespace that supports compression, you can compress these Tables.
- **Supports Serialized Data Writing and Parallel Data Reading**. Only one job can write data to a Table instance at a time. However, multiple jobs can read data from a Table instance, even when another writing job is running in parallel.
- **Does Not Support Logical Rollback on Failure**. In the event of a failure during data load, the system does not roll back the data that is already committed to the database. For example, if a job is inserting 5000 records into a Transactional High Throughput Table, and encounters an error at the 4000th record, the 3999 records already committed to the database are not rolled back.

  This is different from all the other data processing types, in that, when a job fails, the system removes all the data written to the Table and committed to the database as part of that job.
- **Does Not Require Unique/Primary Key**. You do not have to specify a unique/primary key for Table instances of this processing type. However if you specify

these constraints, the Transactional High Throughput data processing type enforces them.

SQL*Loader Processing for SAS Programs

You can improve performance for SAS Programs by directly inserting records into tables using the SQL Loader.

The following conditions are required:

- The Program must be of type SAS Program (not SAS Format or SAS Macro).
- The target tables must be of data processing type Staging, either with or without Audit.
- The target tables must also have the Target as Dataset attribute set to Yes.
- A service in a SAS service location must have the absolute path of the SQL*Loader executable in its Details field (instructions are in the *Oracle Life Sciences Data Hub System Administrator's Guide*).

If all these conditions are not met, the system uses conventional processing.

Using Tables as Pass-Through Views

For Table instances that are the target of an Oracle-technology Load Set, you can specify in the Process Type drop-down list that the Table instance is a pass-through view. No processing type is required because the system does not write data to the Table instance. Instead, you use the Table instance as a view to see data in the source system.

For further information, see About Oracle Tables and Views Load Sets.

Data Auditing, Snapshots and Refresh Groups

To enable auditing of all changes to data, provide consistent views of data at a point in time, and provide consistent views across a set of tables, Oracle LSH implements table auditing, snapshots, and multi-table refresh groups.

- Data Auditing
- Data Snapshots
- Refresh Groups

Data Auditing

When a table is audited, Oracle LSH never truly deletes data from the table, but records each change to each record over time, including deletion. Because the data remains in the table, with timestamps for each update, you can recreate the state of data in the table at any previous point in time in a data snapshot.

If a Table is used in such a way that an audit trail is unnecessary—for example, the Table is used only as a temporary staging area—then you can save space in the database by using a processing type without auditing.

- Auditing in Transactional and Reload Processing

- Auditing in Staging Processing

Auditing in Transactional and Reload Processing

The audit facility for Transactional and Reload processing is based on a self-journaling mechanism: record "versioning" within a table. Each record's uniqueness is defined by a primary or unique key; for example, in a patient enrollment table, the patient ID is the primary key. No other patient has the same ID.

Therefore each time a Program or Load Set writes a row to a Table instance with a patient ID that already exists in the Table instance, the stem sees the new row as an update for an existing patient. Instead of making a change in the existing row, however, the system sets the end timestamp for the existing row for the patient and inserts a new row with the current Column values and a creation timestamp equal to the end timestamp of the previous row. Each row effectively becomes a version of a patient record that is current during the period between its creation timestamp and its end timestamp. Only one version of a record is current at any point in time. The current row always has distant future end timestamp: 3 million Julian.

The timestamp used for records' creation and end timestamp is constant for a given master job; it is the refresh timestamp (REFRESH_TS) of the job.

Insertions: When a new record—a record whose primary or unique key value does not match any other in the Table instance—is inserted, the system sets the creation timestamp to the job timestamp (REFRESH_TS) and the end timestamp to 3 million Julian.

Updates: As records are updated, either through explicit updates in transactional processing or implicitly in reload processing, for each modified record the system:

- sets the end timestamp to the job's refresh timestamp for the most recent row
- inserts a new row with a creation timestamp equal to the job's refresh timestamp and an end timestamp of 3 million Julian

Deletions: When a record is deleted, either explicitly through transactional processing or implicitly through full reload processing, the system sets the end timestamp of the current row to the job's refresh timestamp minus 1 second and also inserts an additional row to explicitly document the deletion. This deletion row has a creation timestamp set to the job's refresh timestamp minus second and an end timestamp of the job's refresh timestamp. This is called "soft-deletion;" the record remains in the database.

Auditing in Staging Processing

Auditing in Staging processing is much simpler; the system saves a copy of data written to the Table instance for each job. Each set of records has the same creation timestamp—the refresh timestamp of the job that created them. With each job, the entire current set of records is inserted. While the job that inserts the records can also perform updates or deletions of those records, there is no audit of these changes.

Data Snapshots

Snapshots allow Oracle LSH to view the data in one or more Table instances in the state it was in at the completion of any job that modified data in the Table instance(s).

The system uses the master job's refresh timestamp for the creation timestamp (and, for deleted records, end timestamp) of all records processed in a master job.

The system uses the master job's refresh timestamp for the creation timestamp (and, for deleted records, end timestamp) of all records processed in a master job, even if the job takes place over a number of hours and includes incremental commits. A master job is any job submitted explicitly for execution. Some master jobs include subjobs; for example, Workflows, Report Sets and any executable submitted using backchaining.

A snapshot comprises all the records in a Table instance that are current at a given point in time. For reload and audited transaction Table instances, a snapshot is the set of records whose end timestamp is greater than, and whose creation timestamp is less than or equal to, a given refresh timestamp. For audited staging Table instances, it is the set of records whose creation timestamp equals the refresh timestamp.

Using snapshots has the following benefits:

- When Programs subsequently access the resulting data, the system bases a stable view of the data on the most recent refresh timestamp, providing a consistent view of the data even if the table is being updated at the time the Program reading the data is running.
- It is possible to recreate data as it was at an earlier point in time.
- Access to incompletely loaded data is prevented and rollbacks of incomplete loads are supported even if there have been incremental commits.

You can label snapshots in two ways:

- When you run a job you can specify a label to be applied to the source and/or target Table instances (see instructions for "Generating Reports and Running Other Jobs" in the *Oracle Life Sciences Data Hub User's Guide*).
- In a Work Area, you can apply a snapshot label to a data timestamp in one or more Table instances in the Work Area; see Adding, Removing, or Moving a Snapshot Label.

Refresh Groups

The system treats any set of tables that are populated by the same master job as a refresh group. The system prevents access to any of the tables until changes to all the tables populated in the Program are complete. The Oracle LSH job tracking record for the execution of each of these applies to all of their contained tables.

For example:

- The target Table instances of a Load Set or Program that populates a multiple table instances
- a Workflow containing multiple Programs that write to table instances

Processing Data Subsets

It is possible to define a Program that processes only a subset of the data in its source tables based on subsetting Parameters. However, if the Program is defined in such a way that the data subset Parameters are modifiable, it does not make sense to use target table processing types that are intended to support incremental or time-based snapshot processing.

Use staging Table instances with or without audit for subsetting. Use staging without audit when you do not need to maintain audited results; for example, in a Workflow that reads data for a subset of patients, reorganizes the data into the temporary table, and runs a series of reports.

If you need to save the results, use staging with audit.

If you want to use Parameters to process different subsets without deleting the data from previously loaded subsets, use reload processing in incremental mode.

You can use staging tables with any Program technology type (SAS, Oracle Reports, or PL/SQL).

Forward Chaining

Using forward chaining execution you can automatically update all tables downstream in a data flow that directly or indirectly read from a table when that source table is updated. When the first Program, Load Set, or Workflow in the forward chain runs, it triggers the execution of some or all Programs or other executables that read from the tables it writes to. Those executables can in turn trigger the execution of all Programs or other executables that read from the tables they write to, and so on.

The system detects dependencies and executes the chain in the proper order so that data in all impacted tables is synchronized and all report outputs reflect the latest data. Where there are no dependencies, jobs run in parallel.

Forward chaining can include executables in many Work Areas, Application Areas, and even Domains—unlike Workflows, which are limited to a single Work Area. All executable object types—Load Sets, Programs, Workflows, Report Sets, and Data Marts—can be part of a forward chaining process. Load Sets can only be triggered manually as the first object in a chain, and Data Marts can only be the end of their data flow branch.

You can trigger a forward chain by using the Forward Chaining-enabled Execution Setup of any executable in the chain, and that will trigger the execution of downstream executables in just that downstream branch—those that directly or indirectly read from the tables written to by the executable object you run.

To participate in a forward chain execution, a Program or other executable object instance and related objects must meet all the following conditions:

- The executable must have an Execution Setup with Forward Chaining enabled.
- If it is not the first object in the chain, the preceding executables in the chain—the ones that write to the tables it reads from— must have an Execution Setup with Cascade enabled.
- The Work Area, Application Area, and Domain(s) containing the executable must have Forward Chaining enabled.
- The executable's validation status must be equal to or greater than its Work Area's Usage Intent.
- The Work Area's own validation status should be equal to or greater than its Usage Intent.

> **Note:**
>
> If the user who submits the forward chain job does not have the privileges to execute any object included in the chain, the system does not execute that object or any objects downstream from it.
>
> If any individual object execution fails, the system does not execute any objects downstream from it. Other subjobs continue unaffected, but the master job status is Failed.
>
> If the system detects a loop in dependencies, the job fails.

By default, all Domains, Application Areas, and Work Areas have Forward Chaining enabled. If it is disabled for a Domain, Application Area, or Work Area, no executables anywhere in the container can participate in forward chaining, even if they meet the other requirements.

The first Execution Setup created for an executable object also has Forward Chaining (and Cascade) enabled by default.

You can change the Forward Chaining and Cascade settings at any level at any time.

Report Only Mode: You can run an executable that qualifies for forward chaining in Order Only mode to produce a report of all the downstream executables that would be executed as part of a forward chain. The report is part of the job's log file.

Starting a Forward Chain Process: When you submit an Execution Setup that has Forward Chaining enabled, select a Data Currency of **Most Current Available (Trigger Forward Chain)**. The system displays the following additional fields:

- **Report Only**: Select it to run in Report Only Mode.
- **Desired Top Level Hierarchy**: By default this is set to All and all possible objects that are part of the forward chain are included. However, you can limit the scope of the job by selecting a Work Area, Application Area, or Domain above the object being executed that you want to serve as the top level of the hierarchy; objects outside it are not included in the job.

Backchaining

In backchaining, the system checks upstream along a data flow where backchaining is enabled to see if more recent data is available. You create a backchain data flow by defining a backchain-enabled Execution Setup for each executable along the data flow, and submitting the Execution Setup for execution.

If the system finds more recent data anywhere in the data flow where backchaining is enabled, it runs executable objects in the data flow starting at the point with more recent data in order to feed the most recent data into the program being executed.

Oracle Clinical Data Extract Load Sets (both Oracle and SAS) allow the system to interpret data currency in the source system so that a backchain can reach as far as the source tables or data sets in Oracle Clinical. If more current data exists in the source system, the backchain process triggers the execution of the Load Set(s) to load the more recent data into Oracle LSH.

- How Backchaining Works

- Backchaining Rules
- Backchaining Tips

How Backchaining Works

When at least one of the Programs or Load Sets that writes data to a Table instance that a Program instance reads from has a backchain Execution Setup defined and submitted, Oracle LSH displays "Most Current Available" as an allowed value for the Data Currency system parameter in the Execution Setup for the Program instance. When a user submits a job with the Data Currency parameter set to Most Current Available, the system invokes a backchain process that does the following:

- The backchain process checks each Program or Load Set instance that writes to the job's source Table instances, looking for a job submission with the Submission Type system parameter set to Backchain. This is a backchain job submission. (To create a backchain job submission, create an Execution Setup with Submission Type set to Backchain and submit it for execution.)
- For each Program instance with a backchain job submission, the backchain process checks each Program or Load Set instance that writes to its source Table instances to see if those Programs or Load Set instances also have a backchain job submission.
- The backchain job continues to look farther and farther upstream each branch of the data flow until it reaches either the source data system or a Program or Load Set instance that does not have a backchain job submission.
- On each branch, when the backchain job has gone as far upstream as it can, it compares the data currency of the source and target data of the last Program or Load Set that has a backchain job submission.
- If the source data is more current than the target data, the backchain process executes the Program or Load Set instance to refresh the target data. The backchain job then executes the next Program to refresh its target data, and so on. Each job submitted by the backchain process uses the blinding and priority system parameter values set in the job submitted by the user with the data currency set to Most Current Data Available.
- If the target data is already as current as the source data, the backchain job does not execute the Program or Load Set instance, but instead checks the source and target data currency of the next Program downstream, and so on. When it finds a Program whose target data is less current than its source data, the backchain job executes that Program instance and each subsequent Program instance.
- When all branches with backchaining enabled have the most current possible data, the backchain job triggers the execution of the original job submitted (with Data Currency set to Most Current Available). See Figure 13-1.

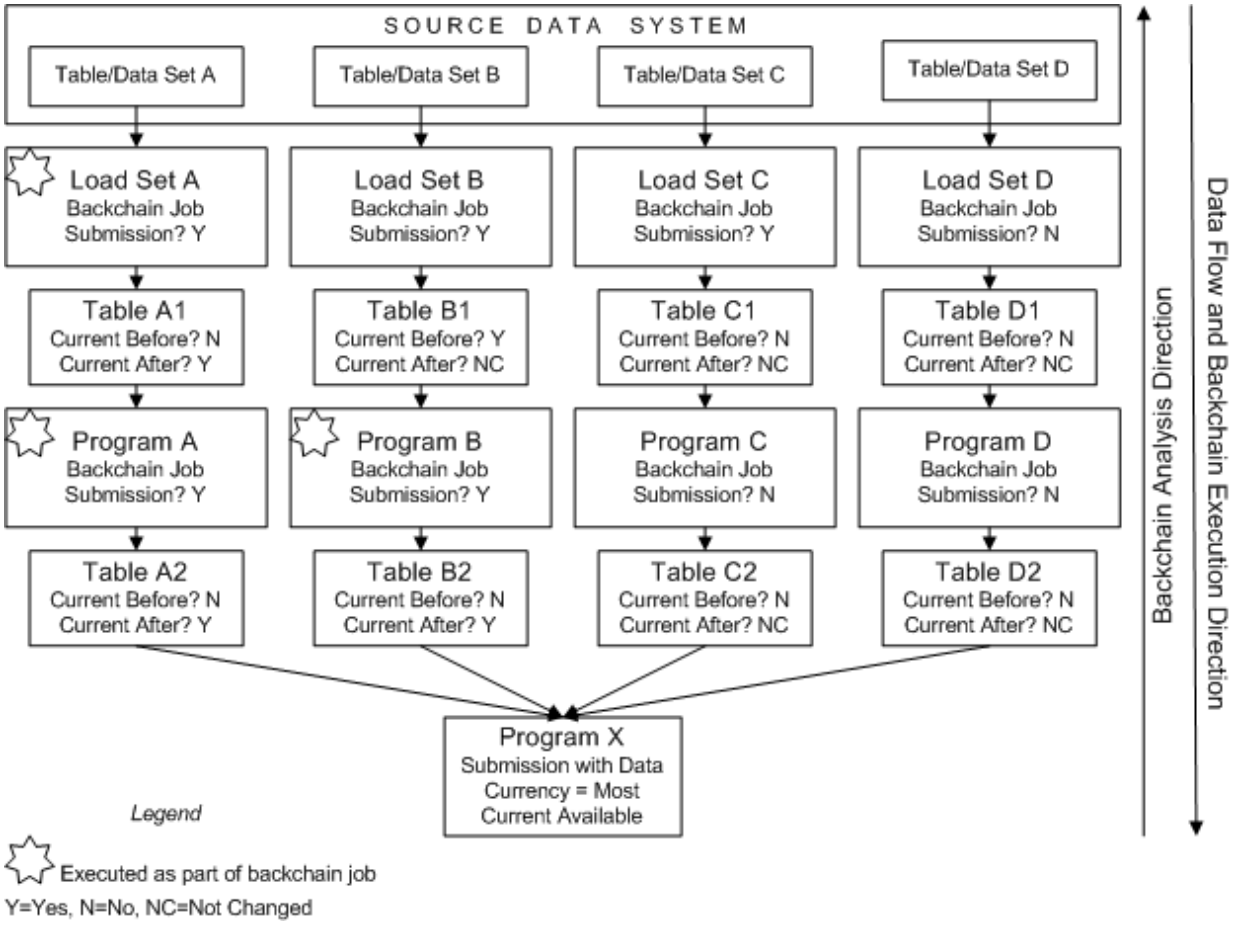
ADDITIONAL INFO: BH: When a backchain process triggers a backchain ES, does it create a new job for it? and display that job in the UI too?

JREES: Yes, a Master Job for that execution setup is started and shows up in Job UI as a normal job. In fact, CDR also creates a 'cloned' version of the backchain submission (see below) for each job that is started, as well. This is because the submission must pick up certain of the system parameters (such as blinding and priority) from the job that initiated the backchain.

BH: Does the backchain logic look for the existence of an ES with a submission type of Backchain, or for the submitted job of submission type Backchain?

JREES: It looks for a Submission with submission type of Backchain. When you submit a job, CDR creates a submission record and a Job record for the first execution of the job (repeating jobs create a new job record for each time they repeat, but only have the one submission).

Figure 13-1 Backchain Example



In the example shown in Figure 13-1, the user submits Program X with the Data Currency system parameter set to Most Current Available. This submission starts a backchain process that checks all the Programs that populate the Table instances that Program X reads from for a backchain job submission. In this example, the backchain process finds that Programs A and B have backchain job submissions, but Programs C and D do not. The backchain process then checks the Load Sets that populate the source Table instances of Programs A and B for a backchain job submission. Load Sets A and B both have backchain job submissions.

The backchain process then checks the data currency of the Load Sets' source and target data. Load Set A's target data is less current than its source, but Load Set B's target data is current. The backchain process checks the data currency of Table B1 compared to Table B2, and finds that B2 is less current.

The backchain process executes Load Set A, and Table A1 becomes current. The backchain process executes Program A, and Table A2 becomes current. The backchain process executes Program B, and Table B2 becomes current. The backchain process then executes Program X, which then has the most current data

available. However, because Programs C and D did not have backchain jobs submitted, Program X does not have the most current data from the source system for those streams. This is true even for stream C, where Load Set C has a backchain job submission but Program C does not.

Note that even if Table A2 is current in relation to its source Table before the backchain process, it becomes noncurrent after the execution of Load Set A updates Table A1. However, Program A then runs and Table A2 becomes current again.

Backchaining Rules

The system enforces the following rules:

- A particular executable can have only one Execution Setup with a submission type of Backchain.
- Only master jobs can have backchaining Execution Setups or run with Most Current Data Available set.
- The same Load Set or other executable with a backchain job submission can be executed by backchain processes initiated by any number of executable objects downstream in a data flow submitted with data currency set to Most Current Data Available. For example, if more than one Program reads from the Tables populated by a Load Set, and if the Load Set has a backchain job submission, the Load Set can be executed as part of a backchain process when either Program is submitted with its data currency set to Most Current Data Available.
- If an executable has multiple source Tables and multiple executables write to those Tables, each of these data flows is processed separately for backchaining, and backchaining can execute successfully even if some data flows go all the way back to the source system and others reach a point where backchaining is not enabled within Oracle LSH.

  For example, if you have a Program that combines treatment codes with patient data, you may not want to enable backchaining on the treatment codes, because they should never be automatically loaded into Oracle LSH, but you do want to see the most recent patient data.
- Backchaining is not possible in Load Sets that load files: Text and SAS.
- Backchaining is not possible with Load Sets whose target Table instances are created as pass-through views. The execution of the entire backchain fails.
- The security for the entire backchain is determined by the security required for the job submitted with Data Currency set to Most Current Available. In other words, if a user has the privileges required to run Program X in the example above, he or she can submit Program X and the entire backchain job can run and invoke every job required for a successful backchain, even if the user does not have the security privileges required to run any of the upstream jobs triggered by the backchain job.

Backchaining Tips

Keep the following in mind:

- Submit the Execution Setup
- Recover from a Canceled Job
- Supply Default Parameter Values

- Do Not Include Workflows with Approval Requests
- Expect Slower Performance

Submit the Execution Setup

When you define a backchain Execution Setup, you must click Submit when you are finished. This does not actually run the job, but the Execution Setup will not run during an actual backchain process unless you have submitted it once already.

Recover from a Canceled Job

If a user cancels the job created for the backchain Execution Setup, the backchain process cannot run that job or any job dependent on it (upstream, or earlier, in the data flow). To recover, do the following:

1. Create another Execution Setup with a submission type of Backchain.
2. Set the Force Duplicate Execution Parameter to Yes.
3. Submit the Execution Setup.

Supply Default Parameter Values

Since backchain jobs are executed automatically, you must supply default values for all required Parameters in backchaining Execution Setups. If you use subsetting Parameters, be sure to be consistent in all the Execution Setups involved in a single backchain job.

Otherwise, if you allow manually submitted executions of the same program to use different subsetting parameter values, then the tables may look up-to-date to backchaining, but only be up-to-date for a particular, different subset of the data.

Do Not Include Workflows with Approval Requests

The system allows you to include a Workflow in a backchaining data flow. However, if a Workflow includes an activity such as an Approval Request that requires manual intervention, the backchain will not be able to complete automatically. Therefore, do not include Workflows in a backchain data flow if they contain Approval Requests.

Expect Slower Performance

Because backchaining execution involves checking for more recent data upstream in the data flow, and then executing multiple objects, processing potentially large amounts of data, it is generally much slower than standard Oracle LSH execution, which operates on the most current data or a specified snapshot of data in the immediate source Table instances.

Managing Blinded Data

For additional information on data blinding, see "Security for Blinded Data" in the *Oracle Life Sciences Data Hub Implementation Guide*.

You may want to hide, or blind, treatment codes or other information that would reveal which patients were receiving which treatments.

Oracle LSH supports data blinding in:

- the Oracle LSH Table instances you specify
- all reports and other outputs generated using one or more blinded Table instances as a source
- all Table instances downstream in the data flow from a blinded Table instance: If a Program instance that reads from a blinded or unblinded Table instance attempts to write data to a nonblinded target Table instance, the submission fails—unless the target Table instance is explicitly authorized to accept data from such a Program and a user with Blind Break privileges explicitly confirms that the Program can be executed.

All Table instances have a Blinding flag attribute that indicates whether or not they may contain data that is sensitive and must be blinded at some point in time.

If a Table instance's Blinding Flag is set to **Yes**, then Oracle LSH maintains two partitioned sets of rows for the Table instance: one set of rows of real data and one set of rows of dummy data. Programs that run on data in these Table instances operate on only one set of data at a time: either the real data or the dummy data.

Table instances also have a Blinding Status attribute. If a Table instance's Blinding flag is set to **Yes**, its Blinding Status can be either **Blinded** or **Unblinded** to indicate the current state of the data. If a Table instance's Blinding flag is set to **No**, its Blinding Status can be either **Not Applicable** (the default) or **Authorized**.

This section includes the following topics:

- Loading Real and Dummy Data
- Managing Blinding Along the Data Flow
- Unblinding Table Instances

Loading Real and Dummy Data

When you load data into Oracle LSH, you must declare the data to be either real or dummy data by setting the Blind Break system Parameter. You cannot load real data and dummy data at the same time. However, only one Program or Load Set can write data to any particular Table instance. Different technologies require different approaches to populating the real and dummy data partitions of blinded Table instances.

> **Note:**
>
> Oracle LSH cannot ascertain whether data in an external system requires blinding or not. You must set up your security system so that only people who understand the issues and the source data can run Load Sets that may load sensitive data.

- SAS and Text Load Sets
- Oracle Tables and Views
- Oracle Clinical Randomization

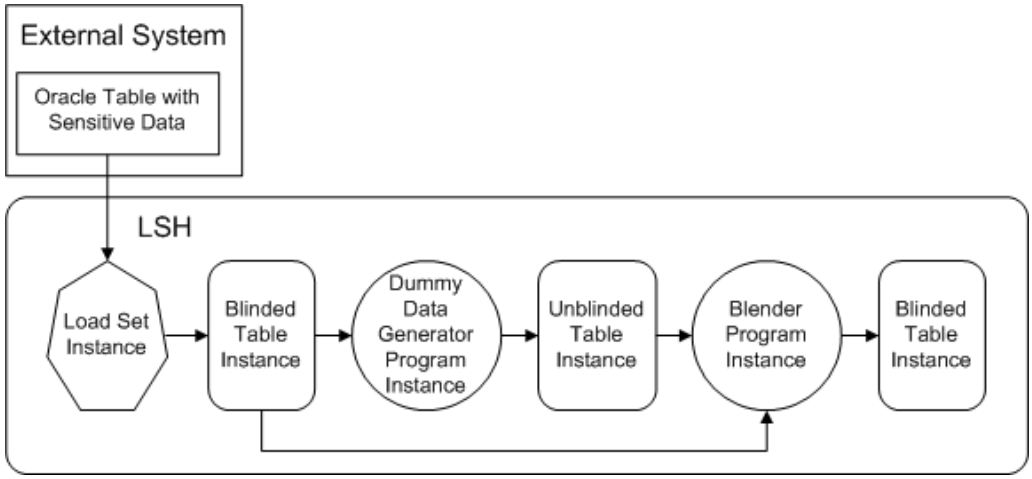
SAS and Text Load Sets

Define your SAS and Text Load Sets so that the file to be loaded must be specified at runtime. Load a file containing real data in one run, and a file with the same structure but containing dummy data in another run. When you load each file, take care to set the Blind Break system Parameter correctly. The system flags each record in the blinded file as blinded and each record in the dummy file as not blinded.

Oracle Tables and Views

The same strategy does not work with Oracle Load Sets because the source table is part of the Load Set definition. Instead, as shown in Figure 13-2, you can:

Figure 13-2 Populating the Blinded and Nonblinded Partitions with Oracle Source Data



- Create a Load Set to load all data from the source table, defining the target Table instance with its Blinding flag set to **Yes** and its Blinding Status set to **Blinded**.
- Create a dummy data generator Program that reads from the Load Set's target Table instance and writes to a nonblinded Table instance with the same structure. The Program should retain the primary key column values but replace the data in all columns that contain sensitive data with dummy data. The target Table instance must have its Blinding flag set to **No** and its Blinding Status set to **Authorized**.
- Create another Program that simply reads data and writes the data—with no changes—into another Table instance with the same structure, but with its Blinding flag set to **Yes** and its Blinding Status set to **Blinded**. Map the Program to two source Table instances: the blinded Load Set target Table instance and the nonblinded target Table instance of the first Program. Use a Parameter to determine which Table instance the Program reads from.

  To load real data into the blinded partition of the target Table instance, run this Program with its Blind Break system Parameter set to **Real (Blind Break)** and the Parameter you created set so that it reads from the blinded Table instance.

To load dummy data into the dummy partition of the target Table instance, run this Program with its Blind Break system Parameter set to **Dummy** and the Parameter you created set so that it reads from the nonblinded Table instance.

> **Note:**
>
> Blind Break privileges are required to run both Programs.

Oracle Clinical Randomization

When you load treatment codes into Oracle LSH from Oracle Clinical using a Randomization Load Set, the privileges you have in Oracle Clinical and the state of the data in Oracle Clinical determine which data you can load into Oracle LSH. Oracle LSH partitions the data appropriately; see Oracle Clinical Randomization .

Managing Blinding Along the Data Flow

When a Load Set writes to a Table instance whose Blinding flag is set to **Yes**, the system changes the values available for the Blind Break system Parameter in the Execution Setup for the Load Set to **Real (Blind Break)** and **Dummy** instead of **Not Applicable**.

The same is true for any Program that reads data from that blinded Table instance and so on downstream in the data flow as Programs read from blinded or unblinded Table instances and write to other blinded or unblinded Table instances.

In each case, the person running the Program must set the Blind Break system Parameter in the Execution Setup.

- If it is set to **Real (Blind Break)**, the system runs the Program on the data in the blinded partition of the source Table instance and writes to the blinded partition of the target Table instance.
- If it is set to **Dummy**, the system runs the Program on the data in the dummy partition of the source Table instance and writes to the dummy partition of the target Table instance.
- If you are using DMW and want to process masking values instead of the real data in LSH, use the **Dummy Blind Break** option. After program execution, DMW masking values are stored in the dummy partition of blinded target tables.

> **Tip:**
>
> Oracle recommends processing DMW data in DMW, not LSH. In particular, do not run DMW transformations and validation checks in LSH.

The Dummy Blind Break option is not available if any of the source tables is blinded at the table level.

Special privileges are required to run a Load Set or Program that has one or more source or target Table instances with its Blinding flag set to **Yes**. For complete

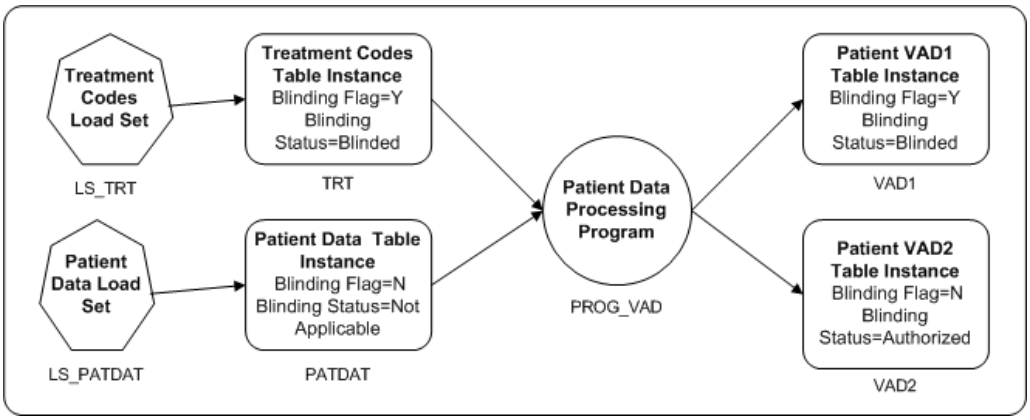
information on blinding-related privileges, see the chapter on security in the *Oracle Life Sciences Data Hub Implementation Guide*.

Normal Usage: Oracle LSH requires that downstream Table instances have the Blinding flag set to **Yes**, but you must set the flag manually. Oracle LSH enforces the rule at runtime. If you attempt to run a Program that writes real data from a Table instance whose Blinding flag is set to **Yes** (with a Blinding Status of either **Blinded** or **Unblinded**) into a nonblinded Table instance (with its Blinding flag set to **No** and its Blinding Status set to **Not Applicable)**) the submission fails.

Exception Authorization: There may be cases where you need to create a Program that reads from one or more blinded or unblinded Table instances and writes to one or more nonblinded Table instances; for examples, see Loading Real and Dummy Data and Figure 13-3 below.

In this case you must set the nonblinded target Table instance's Blinding flag to **No** and its Blinding Status to **Authorized**. The system then allows users with special privileges on the blinded or unblinded source Table instances to run the Program after confirming that it is safe.

Figure 13-3 Reading from a Blinded Table and Writing to a Nonblinded Table



In this example, one source Table instance contains treatment codes and is blinded. The other contains patient data and is not blinded. A Program reads from both Table instances, transforms data, and writes to two Table instances. One target Table instance, VAD1, combines patient data with treatment codes and must be blinded. However, target Table instance VAD2 does not include any treatment code information and does not need to be blinded.

Only users with Blind Break privileges can run PROG_VAD on real data as long as either Table instance TRT or VAD1 has a Blinding Status of Blinded. If the Blinding Status of both of Table instances changes to Unblinded, then users with Read Unblind privileges as well as those with Blind Break privileges can run PROG_VAD.

Unblinding Table Instances

At some point, such as the end of a clinical trial, you may want to make the real, sensitive data that has been blinded during the study, available to a larger group of people for analysis and reporting. To do this, you unblind Table instances that have been blinded, and then run Programs on the real (now unblinded) data.

Special security privileges are required to unblind and reblind a Table instance. See the security chapter in the *Oracle Life Sciences Data Hub Implementation Guide* for information.

To unblind a blinded Table instance:

1. Navigate to the Table instance in its Work Area.
2. Click **Update**.
3. Change the **Blinding Status** to **Unblinded**.

> **Note:**
>
> You can reblind an unblinded Table instance by setting the **Blinding Status** back to **Blinded**.

4. Click **Apply**.

Using Message-Triggered Submission from External Systems

This section contains the following topics:

- About Message-Triggered Submission
- Setup Required
- XML Message Requirements

About Message-Triggered Submission

It is possible to trigger jobs in Oracle LSH by sending an XML message from an external system. For example, if you load patient data into Oracle LSH from Oracle Clinical, you may want to wait until batch validation completes successfully before loading the updated data into Oracle LSH.

Oracle LSH listens at its Event Queue (an Oracle Streams Advanced Queue) for incoming messages. When a message arrives, the system parses the message and triggers the execution of the job.

The message must use a specific XML schema (see Example 13-1). The system checks the validity of the schema structure and all its supplied values. If anything is invalid, the system does not execute the job.

The system sends an email notification to the submitting user of success, failure, warning or error, if the XML message so specifies.

A message cannot exceed a total of 4000 characters. Messages are never deleted from the Event Queue.

For information about creating an Execution Setup to be triggered upon receipt of an XML message, see Creating, Modifying, and Submitting Execution Setups.

Setup Required

To set up message-triggered submission you must do the following:

- **Execution Setup**. Create an Execution Setup for the Load Set (or other executable) that accepts the Triggered submission type (see Creating, Modifying, and Submitting Execution Setups) as well as the Immediate and Deferred submission types.
- **Oracle LSH User Account**. Create a user account in Oracle LSH for the external system user who will create the database link from the remote location. If you want the external system user to receive email notifications, set it up in the user account; see "Creating User Accounts" in the *Oracle Life Sciences Data Hub System Administrator's Guide* for information.

  If the application that triggers the submission is part of the Oracle E-Business Suite, it is not necessary to create another user account.
- **Oracle LSH Database Account**. If the external system's database is separate from Oracle LSH, create a database account in Oracle LSH for the user account; see "Creating Database Accounts" in the *Oracle Life Sciences Data Hub System Administrator's Guide* for information.
- **Grant Execute on the API Security Package cdr_pub_api_initialization** to the user with the LSH database account.
- **Database Link**. In the remote database, use the Oracle LSH database account user ID and password to create a database link to the Oracle LSH database.
- **XML Message**. Send an XML message from the external system using the required XML schema; see Example 13-1. In the XML message, the user ID you specify must be the same as the user ID used to create the database link.

  An Oracle LSH API package called **cdr_pub_exe_msg_api** with the procedure Submit Message is available for use in enqueuing messages; see the *Oracle Life Sciences Data Hub Application Programming Interface Guide*.

  For information about enqueuing messages, see the *Oracle® Streams Advanced Queuing User's Guide* at `http://download.oracle.com/docs/cd/B19306_01/server.102/b14257.pdf`.

XML Message Requirements

You must use XML messages that follow a specific schema (shown in Example 13-1).

The schema requires the following information:

- Executable Specification
- System Parameter Values
- User ID
- Request ID
- Required XML Schema

Executable Specification

The XML message must identify an existing, installed executable object instance to be executed, including its Domain(s), Application Area, and Work Area, and the Execution Setup to be used. The Execution Setup must be active and defined to accept the Triggered submission type.

> **Note:**
>
> If the executable object instance is contained in a Work Area and Application Area that are contained in a nested Domain, enter the names of all the Domains, starting at the top level and inserting a forward slash between Domain names. For example, if the object is contained in Study123_Domain contained in ProjectABC_Domain, enter ProjectABC_Domain/Study123_Domain.

System Parameter Values

The XML message must supply valid values for all System Parameters whose value should be different from the default values defined in the Execution Setup. The System Parameters included in the schema are:

- **Submission Type:** The XML message must supply a submission type of either Immediate or Deferred. If Immediate, the system executes the job immediately after receiving and processing the message. If Deferred, the system executes the job at a date and time specified in the message using the format DD-MON-YYYY HH24:MI (for example, 31-MAY-2010 13:45). If, at submission time, the supplied scheduled time is in the past, the job will be executed immediately.
- **Run Mode:** If you are using the Reload processing type, the XML message must supply a run mode of either Full or Incremental. See Data Processing Types.
- **Execution Priority:** The XML message must supply an execution priority of either Normal, Low, or High.
- **Notification Type:** The XML message must specify under what circumstances the system should send the user a notification. You can specify one or more of the following values:
  - **None**. The system never sends a notification to the user.
  - **Success**. The system sends a notification to the user if the job executes successfully.
  - **Failure**. The system sends a notification to the user if the job execution fails.
  - **Warning**. The system sends a notification to the user if the job ends with a warning.
  - **Error**. The system sends a notification to the user if the job end with an error.
- **Data Currency Type:** The XML message must specify the data currency type of the data, and the Execution Setup must be defined appropriately for that type.
  - **Current**. The system runs the job on data that is current in the immediate source tables at the time of execution.

  - **Backchain**. The system uses backchaining to find the most current data available. The Execution Setup must have a submission type of Immediate or Deferred and a data currency type of Most Current Available. The Execution Setups for Programs feeding data to the source tables must have a submission type of Backchain. See Backchaining for further information.

    > **Note:**
    >
    > The system does not support using message-triggered submission on data snapshots. Do not enter Snapshot as a value.

- **Snapshot Label:** If you want the job to apply a snapshot label to Oracle LSH data, enter one of the following values:
  - **Target**. If set to Target, the system applies a snapshot label to target Table instances only.
  - **Both**. If set to Both, the system applies a snapshot label to both source and target Table instances.
- **Label:** If you enter either Target or Both as the value for Snapshot Label, enter the text of the label as this Parameter value.
- **Blind Break Flag:** The XML message must indicate whether this job is being run on real, blinded data. All Oracle LSH security rules pertaining to blinding apply to message-triggered jobs. The submitting user must have the required privileges or the job does not run.
  - **No**. A value of No indicates that the job is not a blind break. This is the default value.
  - **Yes**. A value of Yes indicates the job is to be run on real, blinded data.
- **Force Execution Flag:** The XML message must indicate whether or not to execute the job even if the resulting output will be the same as for the last execution.
  - **Yes**. If set to Yes, the system always executes the job.
  - **No**. If set to No, the system compares the executable definition version, the Parameter settings, and the source data currency for the previous execution of the same executable with the current ones and does not execute the job if they are all the same. The system sends a Notification to the Home page of the user who submitted the job stating that the job was not executed.

User ID

The Oracle Applications Account ID of the user for whom the job is being submitted:

- If the message is sent from a remote database, the system checks the user ID specified in the message matches a database account in Oracle LSH.
- If the message is sent from within Oracle LSH, the user ID specified in the message must be the same as the logged-in Oracle LSH user who sends the message.
- If the message is sent from an application in the Oracle E-Business Suite that shares the same database as Oracle LSH, the database account is APPS. The system checks that the user ID specified in the message matches the Oracle

Applications Account ID with which the XML message-sending user has logged into the E-Business Suite.

Request ID

The XML message must supply an identifier to be included in execution logs and messages back to the submitter.

Required XML Schema

The XML message must use the schema shown in the following example.

Example 13-1 Required XML Schema for Messages

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SubmissionRequest">
    <xs:annotation>
      <xs:documentation>Oracle LSH submission request submitted via event queue</
xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ProgramSpecification">
          <xs:annotation>
            <xs:documentation>Identifies the program to be executed</
xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:attribute name="domain" type="xs:string" use="required"/>
            <xs:attribute name="applicationArea" type="xs:string" use="required"/>
            <xs:attribute name="workArea" type="xs:string" use="required"/>
            <xs:attribute name="program" type="xs:string" use="required"/>
            <xs:attribute name="executionSetup" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="SystemParameters" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Provides values for required system parameters.
Elements can be omitted; defaults will be supplied during processing.</
xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="SubmissionTypeRc" minOccurs="0">
                <xs:annotation>
                  <xs:documentation>IMMEDIATE or DEFERRED. If DEFERRED, supply a
datetime, in schedStartTs, at which the job is to be executed. DD-MM-YYYY HH24:MI
IF time is in the past, the job will be executed immediately.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:attribute name="value" default="IMMEDIATE">
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="IMMEDIATE"/>
                        <xs:enumeration value="DEFERRED"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>
                  <xs:attribute name="schedStartTs" type="xs:string"/>
```

```
      </xs:complexType>
    </xs:element>
    <xs:element name="RunModeRc" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="value" default="FULL">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FULL"/>
              <xs:enumeration value="INCREMENTAL"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:element name="ExecutionPriorityRc" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="value" default="NORMAL">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="LOW"/>
              <xs:enumeration value="NORMAL"/>
              <xs:enumeration value="HIGH"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:element name="NotificationTypeRc" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="value" default="FAILURE">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="SUCCESS"/>
              <xs:enumeration value="WARNING"/>
              <xs:enumeration value="FAILURE"/>
              <xs:enumeration value="NONE"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:element name="CurrencyTypeRc" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="value" default="CURRENT">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="CURRENT"/>
              <xs:enumeration value="BACKCHAIN"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
  <xs:element name="SnapshotLabel" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="ApplySnapshotLabel" default="BOTH">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="BOTH"/>
              <xs:enumeration value="TARGETS"/>
            </xs:restriction>
```

```
                    </xs:simpleType>
                  </xs:attribute>
                  <xs:attribute name="Label" type="xs:string"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="BlindBreakFlag" minOccurs="0">
                <xs:complexType>
                  <xs:attribute name="value" default="NO">
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="YES"/>
                        <xs:enumeration value="NO"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>
                </xs:complexType>
              </xs:element>
              <xs:element name="ForceExecutionFlag" minOccurs="0">
                <xs:complexType>
                  <xs:attribute name="value" default="NO">
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="YES"/>
                        <xs:enumeration value="NO"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="ProgramParameters" minOccurs="0">
          <xs:annotation>
            <xs:documentation>If the specified program requires parameter values,
they are specified in this element.</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ParameterSetting" minOccurs="0"
maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Each supplied parameter setting must have name
and a value. </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:attribute name="name" type="xs:string" use="required"/>
                  <xs:attribute name="value" type="xs:string" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="userId" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>Oracle Applications account of the user for whom the job
will be executed. </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="requestId" type="xs:string" use="required">
```

```
        <xs:annotation>
          <xs:documentation>An identifer supplied by the application that generates
the request. This ID will be included in execution logs and messages back to the
submitter.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

14

System Reports

The Oracle Life Sciences Data Hub (Oracle LSH) generates reports on demand that provide an audit trail of the activities performed using Oracle LSH and present information on the state of the system's various components.

This chapter contains the following sections:

- Running and Saving System Reports
- Alphabetical Listing of Reports
- Security Reports
- Data Blinding Reports
- Container Reports
- Object Metadata Reports

Running and Saving System Reports

Information on where to run each report is included in the section on the report.

You can run a report only if you have security access to the screen from which it is launched and to the metadata included in the report.

To generate a system report:

1. In the relevant screen, select **Reports** from the **Actions** drop-down and click **Go**. The system displays the reports that are available in your current context.
2. Select the report you want and click **Generate Report**. If the report has parameters, enter values for the parameters. The system generates the report and displays it in PDF format.
3. To open the report in a separate window, click **Export** and then click **Open**.
4. To save the report to your personal computer, do one of the following:
   - Click **Export** and then click **Save**.
   - Click **Save a Copy** in the PDF toolbar.

   You can print the local copy if required.

> **Note:**
>
> After saving the report to your local machine, you can add it as a validation supporting document to a defined object, if appropriate. The report is then stored in Oracle LSH.

Alphabetical Listing of Reports

Oracle LSH provides the following System reports. Click a report name to view more information.

All Instances (Work Area) Report; see Work Area - All Instances Report
All Instances of a Definition Report
Application Area Library Report
Blind Breaks Report
Blinded Table Instances Audit Report
Blinded Table Instances Report
Blinding Rights Report
Cloning Report; see Work Area Cloning Report
Data Mart Instance Report
Data Mart Report
Domain Library Report
Installation History Report; see Work Area Installation History Report
Load Set Instance Report
Load Set Report
Object Validation Report
Object Version History Report
Operations for a Role Report
Program Instance Report
Program Report
Report Set Instance Report
Report Set Report
Security Report; see User Group Assignments Report
Table Instance Report
Table Report
Unblinded Outputs Report
User Group Assignments Report
Users in Group Report
Version History Report; see Work Area Version History Report
Work Area - All Instances Report
Work Area Cloning Report
Work Area Installation History Report
Work Area Version History Report
Workflow Instance Report
Workflow Report

Security Reports

This section contains the following reports:

- Blinding Rights Report
- Operations for a Role Report
- User Group Assignments Report
- Users in Group Report

Blinding Rights Report

The Blinding Rights report lists each user-defined security role that has blinding-related privileges.

The report includes a section for each of the following:

- Read Unblind Rights for Table Instances
- Unblind Rights for Table Instances
- Blind Break Rights for Table Instances
- Read Unblind Rights for Outputs
- Unblind Rights for Outputs
- Blind Break Rights for Outputs

In each section, the reports display all roles that have the relevant operation assigned, and the subtype the assignment applies to.

To run the Blinding Rights report, click the **Security** tab, then click the **Roles** subtab and follow instructions in Running and Saving System Reports.

Operations for a Role Report

The Operations for a Role report displays all the operations associated with a user-defined security role in Oracle LSH.

For each operation, the report displays the following information:

- Subtype
- Object Type
- Operation
- Timestamp of Assignment of Operation

To run the Operations for a Role report, click the **Security** tab, then click the **Roles** subtab and follow instructions in Running and Saving System Reports.

User Group Assignments Report

The User Group Assignments report displays information on every user group assigned to a particular object. You can choose to include only current assignments or all past assignments.

The following information is displayed for each User Group:

- Group Name
- Activity Type (Inherited, Assigned, Revoked, Unassigned, or Unrevoked)
- Activity Timestamp
- Group Assigned By

To run the User Group Assignments report, navigate to the Properties screen of the Domain, Application Area, Work Area, or object definition or instance whose user

group assignments you want to see, and follow instructions in Running and Saving System Reports.

See Header Information for Object Instances for details on the header section.

Users in Group Report

The Users in Group report lists the roles assigned to a selected user group as well as the users assigned to a particular role. For each role and user, the report displays the following information:

- Role
- Description of Role
- User ID
- Timestamp of Assignment
- Added by
- Timestamp of Removal
- Removed by

To run the Users in Group report, click the **Security** tab, then click the **User Groups** subtab and follow instructions in Running and Saving System Reports. If you want to include users added or removed in the past, select the corresponding checkbox.

Data Blinding Reports

This section contains the following reports:

- Blind Breaks Report
- Blinded Table Instances Audit Report
- Blinded Table Instances Report
- Unblinded Outputs Report

Blind Breaks Report

The Blind Breaks report displays information about each time real data has been accessed from a Table instance, including:

- Each time a job has been run against a selected blinded Table instance in Blind Break mode to display real data
- Each time an IDE (integrated development environment such as SAS) has been launched with the Blind Break setting set to **Real (Blind Break)**
- Each time a visualization (through an external tool such as Oracle Business Intelligence Enterprise Edition) has been launched with the Blind Break setting set to **Real (Blind Break)**
- Each time data has been browsed with the Blind Break setting set to **Real (Blind Break)**

The Blind Breaks report displays the following details:

- **Header Information**—Details of Containership, Instances and Definitions. See Header Information for Object Instances for details on the header section.
- **Blind Breaks**—Job ID or Activity Type, Blind break TimeStamp, Blind Break By, Instance Name, Instance Type, Instance Validation Status

To run the Blind Breaks (or Blindbreak) report, navigate to the table instance and follow instructions in Running and Saving System Reports.

Blinded Table Instances Audit Report

The Blinded Table Instances Audit report displays all Table instances whose blinding flag or blinding status has been changed in a particular Work Area. For each such Table instance, the report details every change to the blinding flag or blinding status. For each change, the report displays the following information:

- **Containership**—Domain, Application Area Name, Work Area Name
- **Table Instance Details**—Table Instance Name, Version, Validation Status, Action Type, Timestamp, Action By

To run a Blinded Table Instances Audit report, navigate to the Work Area's Properties screen and follow instructions in Running and Saving System Reports.

Blinded Table Instances Report

The Blinded Table Instances report displays details of all Table instances in a Work Area, regardless of their Blinding Status, and displays their Blinding Flag setting and Blinding Status.

The report has the following information for each Table instance:

- **Containership**—Domain, Application Area Name and Work Area Name
- **Table Instance Details**—Table Instance Name, Version, Validation Status, Source Definition Name, Source Definition Version, Created Time Stamp, Blinding Flag, and Blinding Status.

To run a Blinded Table Instances report, navigate to the Work Area's Properties screen and follow instructions in Running and Saving System Reports.

Unblinded Outputs Report

The Unblinded Outputs report displays all jobs run on real, unblinded data in a particular Table instance.

The Unblinded Outputs report has the following sections:

- **Header Information**—Details of Containership, Instances and Definitions. See Header Information for Object Instances for details on the header section.
- **Jobs and Output Details**—The report displays the following details for each job: Job ID, Job Submission Timestamp, Job Submitted by, Output Name, Output Creation Timestamp, Owning Executable Instance, Executable Type, Executable Validation Status.
- **Table Instance Status**—The report displays the following details for the source Table instance at the time of job submission: Job ID, Validation Status, Blinding Flag, Blinding Status.

To run an Unblinded Outputs report, navigate to the table instance's Properties screen and follow instructions in Running and Saving System Reports.

Container Reports

This section contains the following reports:

- Application Area Library Report
- Domain Library Report
- Work Area - All Instances Report
- Work Area Cloning Report
- Work Area Installation History Report
- Work Area Version History Report

Application Area Library Report

The Application Area Library report displays information on objects contained in a particular Application Area. You can select the types of objects to include in the report. The report lists objects by their type.Besides information on Domain, Application Area Name and Work Area Name, the report lists the following for each object:

- Name
- Description
- Version
- Status
- Checked In or Out?
- Check out by
- Validation Status

To run an Application Area Library report, navigate to the application area's Properties screen and follow instructions in Running and Saving System Reports.

Domain Library Report

The Domain Library report displays information on objects contained in a particular Domain. You can select the types of objects to include in the report. The report lists objects by their type.

For each object, the report displays the Domain name and Classification in addition to the following information:

- Name
- Description
- Version
- Status
- Checked In or Out?
- Check out by

- Validation Status

To run a Domain Library report, navigate to the domains's Properties screen and follow instructions in Running and Saving System Reports.

Work Area - All Instances Report

The Work Area - All Instances report details every object instance in a selected work area.

The report provides Containership information such as Domain Name, Application Area Name, Work Area Name, Validation Status, and Classification. It also contains the following information on every instance:

Instance Details

- Instance Name
- Instance Version
- Last Modified by
- Last Modified TS
- Source Definition Name
- Source Definition Version
- Checked In or Out?
- Definition's Application Area
- Definition's Domain

To run a Work Area-All Instances report, navigate to the work area's Properties screen and follow instructions in Running and Saving System Reports.

Work Area Cloning Report

The Work Area Cloning report details the complete cloning history of a selected work area.

The report provides Containership information such as Domain Name, Application Area Name, Work Area Name, Validation Status, and Classification. It also contains the following information on every cloned work area:

- Work Area Details

Work Area Details

- Clone Work Area Name
- Description
- Status
- Validation Status
- Usage Intent
- Cloned from Work Area Version
- Cloned Timestamp

- Clone Labels (if any)
- Cloned by
- Version Label

To run a Work Area Cloning report, navigate to the work area's Properties screen and follow instructions in Running and Saving System Reports.

Work Area Installation History Report

The Work Area Installation History report is a record of every installation in the selected Work Area.

The report provides Containership information such as Domain Name, Application Area Name, Work Area Name, Validation Status, and Classification. It also contains the following information for every installation:

- Installation Details

Installation Details

- Installation Attempt
- Install Status
- Install Mode
- Force to regenerate scripts
- Installed By
- Installation Timestamp

To run a Work Area Installation History report, navigate to the work area's Properties screen and follow instructions in Running and Saving System Reports.

Work Area Version History Report

The Work Area Version History report is a record of every version of the selected Work Area.

The report provides Containership information such as Domain Name, Application Area Name, Work Area Name, Validation Status, and Classification. It also provides the following information for every version of the work area:

- Version Details

Version Details

- Version
- Description
- Status
- Validation Status
- Usage Intent
- Version Label

- Last Modified Timestamp
- Last Modified By
- Cloned from Version

To run a Work Area Version History report, navigate to the work area's Properties screen and follow instructions in Running and Saving System Reports.

Object Metadata Reports

Oracle LSH object metadata reports present an overview of an object's history and current status.

This section contains the following:

- Common Header Information
- All Instances of a Definition Report
- Data Mart Report
- Data Mart Instance Report
- Load Set Report
- Load Set Instance Report
- Object Validation Report
- Object Version History Report
- Program Report
- Program Instance Report
- Report Set Report
- Report Set Instance Report
- Table Report
- Table Instance Report
- Workflow Report
- Workflow Instance Report

Common Header Information

This section contains the following:

- Header Information for Object Definitions
- Header Information for Object Instances

Header Information for Object Definitions

Header information for object definitions is divided into two parts:

- **Containership**—Domain Name and Application Area Name.
- **Definition Details**—Name, Description, Latest Version, Creation Time, Created by, Checked in or out?, Last Modified Time, Last Modified by, Validation Status, Classification

Header Information for Object Instances

Header information for object instances is divided into three parts:

- **Containership**—Domain Name, Application Area Name and Work Area Name.
- **Instance Details**—Instance Name, Description, Version, Last Modified Time, Last Modified by, Validation Status, Classification.
- **Definition Details**—Definition Name, Description, Latest Version, Checked in or out?, Checked out by, Last Modified Time, Last Modified by.

All Instances of a Definition Report

The All Instances of a Definition report contains information on all the object instances that are based on or point to a selected definition.

The report displays information in the format described in Header Information for Object Instances.

To run an All Instances of a Definition report, navigate to the definition's Properties screen and follow instructions in Running and Saving System Reports.

Data Mart Report

A Data Mart report contains information about the current status of a single Data Mart definition. This is the same information that is available in the Data Mart's Properties screen and its subtabs.

For information on a Data Mart's version history, see Object Version History Report . For information on Data Marts, see Defining Data Marts.

The Data Mart report has three sections:

- **Header**—See Header Information for Object Definitions for details on the Header section.
- **Attributes**—A table displaying the values of the attributes Data Mart Type, File Name, File Name Extension, and Mode.
- **Child Objects**—Tables displaying information about each of the following Data Mart components:

> **Note:**
>
> These tables display all possible information for each component. Some of the table columns may not be applicable to a particular type of Data Mart or to a particular component.

  - **Table Descriptors**. For each Table Descriptor, the report displays the Name, Yes or No settings for Is Target?, Oracle Name, SAS Name, and SAS Library Name.

> **Note:**
>
> All Data Mart Table Descriptors are of type Source, not Target.

  - **Parameters**. For each Parameter, the report displays the Prompt, Default Value, and Yes or No settings for Visible?, and Required?.
  - **Planned Output**. For each Planned Output, the report displays the Name, File Name, Yes or No settings for Primary?, Error if generated?, and Error if not generated?.

To generate a Data Mart report, navigate to the Data Mart's Properties screen. Select **Definition Report** from the list of reports available for object instances while following instructions in Running and Saving System Reports.

Data Mart Instance Report

The Data Mart Instance report includes all the definitional information that pertains to the instance, including information about the underlying Data Mart definition. See Data Mart Report for definition related information.

The Data Mart Instance report contains the information available in the Data Mart instance's Properties screen and its subtabs. It has four sections:

- **Header**—See Header Information for Object Instances for details on the header section.
- **Attributes**—A table displaying values of the attributes Data Mart Type, File Name, File Name Extension, and Mode.
- **Child Objects**—Tables displaying information about each of the Data Mart components- Table Descriptors, Parameters, Planned Output. See Data Mart Report for details of these components. Additionally, you can see mapping information about Table Descriptors.
- **Mapping Information**—Detailed mapping information for the Data Mart instance. This section has two subsections.
  - The first subsection displays the following information: TD Table Definition Name, Description, Latest Version, Checked In or Out, Checked out by, Last Modified Time, Last Modified by, Validation Status.
  - The second subsection displays details about the tables that the Table Descriptors are mapped to. This table has two parts, one for the Table Descriptor and the other for the Table instance.
    * **Table Descriptor**—Consisting of the Column Name and Data Type.
    * **Table Instance**—Consisting of the Column Name, Format String, Default Value, Data Type and Mapping Status.

To generate a Data Mart Instance report, navigate to the Data Mart instance's Properties screen. Select **Definition Report** from the list of reports available for object instances while following instructions in Running and Saving System Reports.

Load Set Report

A Load Set report contains information about the current status of a single Load Set definition. This is the same information that is available in the Load Set's Properties screen and its subtabs. For information on a Load Set's version history, see Object Version History Report .

For information on Load Sets, see Defining Load Sets

The Load Set report has three sections:

- **Header**—See Header Information for Object Definitions for details on the Header section.
- **Attributes**—The second section displays the values of the attributes Adapter Type, Remote Location Name, Database Schema, Study Name, View Type, Design Sub-System, and Yes or No settings for Save to File/Save Input File?.
- **Child Objects**—The third section displays tables with information about each of the following Load Set components.

> **Note:**
>
> Child Object tables display all possible information for each component. Some of the table columns may not be applicable to a particular type of Load Set or to a particular component.

  - **Table Descriptors**. For each Table Descriptor, the report displays the Name, Yes or No settings for Is Target?, Oracle Name, and SAS Name.
  - **Parameters**. For each Parameter, the report displays the Prompt, Default Value, and Yes or No settings for Visible?, and Required?.

To generate a Load Set report, navigate to the Load Set's Properties screen. Select **Definition Report** from the list of reports available for object instances while following instructions in Running and Saving System Reports.

Load Set Instance Report

The Load Set Instance report includes all the definitional information pertaining to the instance, including information about the underlying Load Set definition.

The Load Set Instance report contains the same information available in the Load Set instance's Properties screen and its subtabs. It has four sections.

- **Header**—See Header Information for Object Instances for details on the header section.
- **Attributes**—The second section displays a table with values of the attributes Database Schema, Study Name, View Type, Design Sub-System, and Yes or No settings for Save to File/Save Input File?.
- **Child Objects**— Similar to the Load Set report, the third section displays tables with information about Table Descriptors and Parameters. See Load Set Report for details on Child Objects.

- **Mapping Information**—Similar to the Data Mart Instance report, the last section presents mapping information for the Load Set instance in two subsections. See Data Mart Instance Report for details.

To generate a Load Set Instance report, navigate to the Load Set Instance's Properties screen. Select **Definition Report** from the list of reports available for object instances while following instructions in Running and Saving System Reports.

Object Validation Report

The Object Validation report displays information about the validation life cycle of a selected object instance. The report reflects changes to the object instance's validation status and lists all supporting documents and supporting outputs associated with the object instance.

The report has the following sections:

- **Header**—See Header Information for Object Instances for details on the header section.
- **Validation Status**—For each change in the validation cycle, the report displays the following: Validation Status, Status Update Timestamp, Updated by, and Comments.
- **Supporting Document**—For each supporting document, the report displays the following information: Name, File Name, Description, Version, Uploaded by, Status, Validation Status at Time of Upload.
- **Supporting Outputs**—For each supporting output, the report displays the following information: Output Name, Job ID, Description, Added by, Status, Validation Status at Time of Adding.

To generate an Object Validation report, navigate to the object instance's Properties scree. Select **Validation Report** from the list of reports available for object instances while following instructions in Running and Saving System Reports.

Object Version History Report

The Object Version History report is a record of every version of the selected object definition.

The Object Version History report has two sections.

- **Header**—See Header Information for Object Definitions for details on the Header section.
- **Version History**—The second section displays the following details about the object: Version, Description, Status, Validation Status, Copied from Version, Version Label, Last Modified Timestamp, Last Modified by, Check in/Check out comments, if any.

To generate an Object Validation report, navigate to the Properties screen of an object instance definition. Select **Version History Report** from the list of reports available for object instances while following instructions in Running and Saving System Reports.

Program Report

A Program report contains information about a single Program definition. For information on a Program's version history, see Object Version History Report.

For information on Programs, see Defining Programs.

The Program report contains the same information that is present in the Program's Properties screen and its subtabs. It has three sections:

- **Header**—See Header Information for Object Definitions for details on the Header section.
- **Attributes**—The second section displays a table with the value of the attribute Program Type.
- **Child Objects**—The third section displays tables with information about each of the following Program components:

> **Note:**
>
> Child Object tables display all possible information for each component. However, some of the table columns may not be applicable to particular types of Programs or components.

  - **Table Descriptors**. For each Table Descriptor, the report displays the Name, Yes or No settings for Is Target?, Oracle Name, and SAS Name.
  - **Parameters**. For each Parameter, the report displays the Prompt, Default Value, and Yes or No settings for Visible?, and Required?.
  - **Planned Output**. For each Planned Output, the report displays the Name, File Name, Yes or No settings for Primary?, Error if generated?, and Error if not generated?.
  - **Source Code**. For each Source Code, the report displays the Source Code Name, Description, Yes or No settings for Primary?, Share Type, File Type, Yes or No settings for Sharable?, File Name, Source Code Shared From?, and the actual source code.

To generate a Program report, navigate to the Program's Properties screen. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

Program Instance Report

The Program Instance report includes the definitional information pertaining to the instance, including information about the underlying Program definition. See Program Report for details on Program definitions.

The Program Instance report contains the same information available in the Program instance's Properties screen and its subtabs. It has four sections:

- **Header**—See Header Information for Object Instances for details on the header section.
- **Attributes**—The second section displays a table with the value of the attribute Program Type.
- **Child Objects**—Similar to the Program report, the third section displays tables with information about the following Program components—Table Descriptors, Parameters, Planned Output and Source Code. See Program Report for details.

- **Mapping Information**— Similar to the mapping information of the Data Mart Instance report, the last section presents mapping information for the Program instance in two subsections. See Data Mart Instance Report for details.

To generate a Program Instance report, navigate to the Properties screen of a Program Instance. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

Report Set Report

The Report Set report contains information about the current status of a single Report Set including its child objects. The report contains the same information that is available in the Report Set's Properties screen and its subtabs. You can choose whether or not to include information on Post-Processing and Overlay Template Parameters for the Report Set and each Report Set Entry.

For information on a Report Set's version history, see Object Version History Report.

For more information on Report Sets, see Defining Report Sets.

The Report Set report has following sections:

- **Header**—See Header Information for Object Definitions for details on the Header section.
- **Attributes**—The second section displays Attribute values- Title, Yes or No settings for Strict Numbering?, and Yes or No settings for Unique Numbering?.

  If you selected Post-Processing Parameters or Overlay Template Parameters, the report displays these both at the Report Set level and for each Report Set Entry (see below).
- **Child Objects**—The third section displays tables with information about the following components.
  - **Planned Output**. For each Planned Output, the report displays Name, File Name, Yes or No settings for Primary?, Yes or No settings for Error if generated?, and Error if not generated?.
  - **Post-Processing**. For Post-processing, the report displays Parameter, Yes or No settings for Default?, Yes or No settings for Value Propagation?, Yes or No settings for Visible?, and Yes or No settings for Required?.
  - **Overlay Templates**. For each Overlay Template, the report displays Prompt, Yes or No settings for Default?, Yes or No settings for Read Only?, Yes or No settings for Visible?, and Yes or No settings for Required?.
  - **Parameters**. For each Parameter, the report displays Prompt, Yes or No settings for Default?, Yes or No settings for Read Only?, Yes or No settings for Visible?, and Yes or No settings for Required?.
- Report Set Entries
- Program Instance Under Report Set Entry (Full Title)

Report Set Entries

The Report Set report also displays tables with information about each Report Set Entry in numerical order.

- **Report Set Entry Number**. For the Report Set Entry identified by its number and title, the report displays the Entry Name, Title, Yes or No settings for Pre Narrative?, Yes or No settings for Post Narrative?, Placeholder, Yes or No settings for Has Children?, and Yes or No settings for Omitted?.

  The Pre and Post narratives are also shown with the Report Set Entry's details.
- **Post-Processing**. For Post-processing, the report displays Parameter, Yes or No settings for Default?, Yes or No settings for Value Propagation?, Yes or No settings for Visible?, and Yes or No settings for Required?.
- **Overlay Templates**. For each Overlay Template, the report displays Prompt, Yes or No settings for Default?, Yes or No settings for Read Only?, Yes or No settings for Visible?, and Yes or No settings for Required?.
- **Parameters**. For each Parameter, the report displays Prompt, Yes or No settings for Default?, Yes or No settings for Read Only?, Yes or No settings for Visible?, and Yes or No settings for Required?.

Program Instance Under Report Set Entry (Full Title)

When a Report Set Entry within a Report Set contains a Program instance, the report details the Program definition that the particular Program instance points to.

- **Table Descriptors**. For each Table Descriptor, the report displays the Name, Yes or No settings for Is Target?, Oracle Name, SAS Name, SAS Library Name, Mapping Status, Mapped to Table Instance, TI (Table Instance) Work Area, TI (Table Instance) Application Area, TI (Table Instance) Domain.
- **Parameters**. For each Program instance Parameter, the report displays the Prompt, Default Value, Yes or No settings for Visible?, and Yes or No settings for Required?.
- **Planned Output**. For each Planned Output, the report displays the Name, File Name, Yes or No settings for Error if generated?, and Error if not generated?.
- **Source Code**. For Source Code, the report displays the Name, Description, Yes or No settings for Primary?, Share Type, File Type, Yes or No settings for Sharable?, File Name, Source Code Shared From location and the actual source code.

To generate a Report Set report, navigate to the Properties screen of a Report Set. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

You can select any or all of the following optional components to include in the Report Set Definition report—Post Processing Parameters, Overlay Templates Parameters, Narratives.

Report Set Instance Report

The Report Set Instance report includes all the definitional information that pertains to the instance, including information about the underlying Report Set definition. See Report Set Report.

The Report Set instance report contains the same information that is available in the Report Set instance's Properties screen and its subtabs. It has three sections.

- **Header**—See Header Information for Object Instances for details on the header section.

- **Attributes**—The second section displays a table with values of the attributes Title, Strict Numbering, and Unique Numbering.
- **Child Objects**—Similar to the Report Set report, the third section displays tables with information about each of the following components—Planned Output, Post-Processing, Overlay Templates and Parameters. See the section on Child Objects in the Report Set Report for details. Additionally, you can see mapping information about Table Descriptors.
- Report Set Entries
- Program Instance Under Report Set Entry (Full Title)

Report Set Entries

Similar to the Report Set report, the Report Set Instance report also displays tables with information about each Report Set Entry in numerical order. See the section on Report Set Entries in the Report Set Report for details.

Program Instance Under Report Set Entry (Full Title)

When a Report Set Entry within a Report Set contains a Program instance, the report details the Program definition that the particular Program instance points to.

The Program Instance details are displayed in the following subsections—Table Descriptors, Parameters, Planned Output, Source Code and Mapping Details. See the section on Program Instance under Report Set Entry in the Report Set Report for details.

Mapping Details provides information about the tables mapped with the Table Descriptors of the Program Instance in two parts:

- **Table Descriptor**—Column Name, Data Type
- **Table Instance**—Column Name, Format String, Default Value, Data Type, Mapping Status

To generate a Report Set Instance report, navigate to the Properties screen of a Report Set Instance. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

You can select any or all of the following optional components to include in the Report Set Instance report—Post Processing Parameters, Overlay Templates Parameters, Narratives.

Table Report

A Table report contains information about the current status of a single Table definition. This is the same information that is available in the Table's Properties screen and its subtabs.

For information on Tables, see Defining Tables.

The Table report has the following three sections:

- **Header**—See Header Information for Object Definitions for details on the Header section.

- **Attributes**—The second section displays the values of the attributes Oracle Name, SAS Name, SAS Label, Process Type, Allow Snapshot?, Blinding Flag.
- **Child Objects**—The third section displays tables with information about each of the following Table components:

> **Note:**
>
> These tables display all possible information for each component. Some of the columns may not be applicable to a particular type of Table or to a particular component.

  - **Columns**. For each Column, the report displays the Name, Position, Oracle Name, SAS Name, SAS Label, Data Type, Length, Precision, SAS Format, Default Value, Mandatory?, and Variable Name.
  - **Constraints**. For each Constraint, the report displays the Name, Type, Column Name (s), Join Table, and Value(s) allowed.

To generate a Table report, navigate to the Properties screen of a Table. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

Table Instance Report

The Table Instance report includes all the definitional information that pertains to the instance, including information about the underlying Table definition. See Table Report.

The Table Instance report contains the same information that is available in the Table instance's Properties screen and its subtabs. Similar to the Table report, the Table Instance report has three sections—Header, Attributes and Child Objects. See Table Report for details.

To generate a Table Instance report, navigate to the Properties screen of a Table Instance. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

Workflow Report

The Workflow report contains information about the current status of a single Workflow. This is the same information that is available in the Workflow's Properties screen and its subtabs. For information on a Workflow's version history, see Object Version History Report.

For more information on Workflows, see Defining Workflows.

The Workflow report has the following two sections:

- **Header**—See Header Information for Object Definitions for details on the Header section.
- **Child Objects**—The third section displays tables with information about each of the following Workflow components:

> **Note:**
>
> These tables display all possible information for each component. Some of the table columns may not be applicable to a particular type of Workflow or to a particular component.

- **Executable Instances and Workflow Structures**. For each Executable Instance and Workflow Structure, the report displays the Name, Type, Description, Checked out by, and Status.
- **Parameters**. For each Parameter, the report displays the Prompt, Default Value, Yes or No settings for Visible?, and Yes or No settings for Required?.
- **Transitions**. For each Transition, the report displays From Object Name, From Object Type, Condition, To Object, and To Object Type.
- **Mapping Details**. The report displays information about the mapping between executable instances contained in the Workflow and Table instances.

The Workflow report includes information on each executable object instance included in the Workflow. These may include Data Marts, Programs, Load Sets and Report Sets. The information displayed is the same as in the report for each object type. See Data Mart Report, Program Report, Load Set Report, and Report Set Report.

To generate a Workflow report, navigate to the Properties screen of a Workflow. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

Workflow Instance Report

The Workflow Instance report includes all the definitional information that pertains to the instance, including information about the underlying Workflow definition. The Workflow instance report contains the same information that is available in the Workflow instance's Properties screen and its subtabs.

Similar to the Workflow report, the Workflow Instance report has two sections—Header and Child Objects. See Workflow Report for details.

The Workflow Instance report contains information on each executable object instance included in the Workflow. The object instances may include Data Marts, Programs, Load Sets and Report Sets. The information displayed is the same as in the report for each object type. See Data Mart Instance Report, Program Instance Report, Load Set Instance Report, and Report Set Instance Report.

To generate a Workflow Instance report, navigate to the Properties screen of a WorkflowInstance. Select **Definition Report** from the list of reports available while following instructions in Running and Saving System Reports.

A

Installation Requirements for Each Object Type

Most definitional objects have a status with two possible values: Installable and Non Installable. Some of these objects—object definitions and component objects—are never installed directly, but their status affects the status of other objects. If an object definition is not installable, instances of that definition are also not installable. If a component object is not installable, the object definition of which it is a part may also be noninstallable.

Different object types can be noninstallable for different reasons. This section gives the reasons each object type can be noninstallable. Objects are listed in alphabetical order within their section.

This section contains the following topics:

- Installable Instances and their Definitions
- Component Object Types
- Organizational Objects

Installable Instances and their Definitions

When you install a Work Area, you install the object instances within it and through them, the definitional metadata contained in their source definition objects. Therefore both object definitions and instances have an installation status.

- **Business Area:** A Business Area definition is not installable if The Business Area has no Table Descriptors.
- **Business Area Instance:** A Business Area instance is not installable if:
  - Its source Business Area definition is not installable
  - Any of its Table Descriptor mappings are incomplete
- **Data Mart:** A Data Mart definition is not installable if:
  - It has no Table Descriptors
  - It has any required Parameters with no value set
  - It has any Parameter values that fail validation (either List of Values or programmatic validation)
  - It does not have a value for the Filename attribute
  - The File Name value does not have an extension of .dmp
- **Data Mark Instance:** A Data Mart instance is not installable if:
  - Its source Data Mart definition is not installable
  - Any of its Table Descriptor mappings are incomplete
- **Load Set:** A Load Set definition is not installable if it has no Table Descriptors

- **Load Set Instance:** A Load Set instance is not installable if:
  - Its source Load Set definition is not installable
  - Any of its Table Descriptor mappings are incomplete
- **Program:** Program definitions have an additional installation status called "Installable IDE." A Program definition has this status when it has at least one Table Descriptor and all its Table Descriptors are mapped. When a Program has this status, you can install it (from the Program instance screen only, not the Work Area) and view the data in the Table instances mapped to the Program's source Table Descriptors from the IDE. However, you cannot run the Program in Oracle LSH until you have uploaded the source code and installed the Program instance.

  A Program definition is not installable if:
  - The Program contains a Source Code definition that does not contain a source code file
  - The Program contains a Source Code instance whose definition is not installable
  - The Program contains a Table Descriptor whose source Table definition is not installable
- **Program Instance:** A Program instance is not installable if:
  - Its source Program definition's status is Not Installable or "Installable IDE"
  - Any of its Table Descriptor mappings are incomplete
- **Report Set:** A Report Set definition is not installable if:
  - Parameter instances owned directly by the Report Set or Report Set Entries are not set up to propagate their values to at least one Parameter within a Report Set Entry Program instance
  - Circular Parameter value propagation is defined; for example, the value of an output Parameter of Program instance A is propagated to an input Parameter of Program instance B, and the value of an output Parameter of Program B is propagated to an input Parameter of Program A
  - It contains a Program instance whose status is either Not Installable or Installable IDE.
- **Report Set Instance:** A Report Set instance is not installable if:
  - Its source Report Set definition is not installable or "Installable IDE"
  - Any of the Table Descriptor mappings owned by Program instances within the Report Set instance are incomplete
- **Table:** A Table definition is not installable if it contains no Columns.
- **Table Instance:** A Table instance is not installable if its source Table definition is not installable.
- **Workflow:** A Workflow definition is not installable if:
  - The Workflow does not contain one and only one Start Structure and at least one End-type Structure (End_Success, End_Failure, or End_Warning)
  - Each branch of the Workflow does not end with an End-type Structure
  - All of the Workflow's Transitions do not have an activity at each end

  - Each Workflow activity (except Start and End) do not have an activity at each end
  - Parameter instances owned directly by the Workflow are not set up to propagate their value to at least one Parameter within a Workflow activity
  - Parameter linking conflicts with the execution order of the Workflow; for example, if the output Parameter of a Program is linked to the input Parameter of another Program that is set to execute earlier in the Workflow
  - Transitions are circular; that is, a Transition cannot lead from one activity to another that is set to execute earlier in the Workflow. For example, if you define a Transition from Program A to Program B, and another Transition from Program B to Program C, you cannot also define a Transition from Program C to Program A. The system generates an installation status of Not Installable.

    The exception to this rule is that a Transition following a Notification can point to an earlier activity in the Workflow, if there is an End activity defined. For example, a rejection of an Approval-type Notification can lead to a Load Set earlier in the Workflow, so that the Workflow is effectively restarted from that point. The system generates an installation status of: Installable, Includes Circular Reference.

- **Workflow Instance:** A Workflow is not installable if:
  - Its source Workflow definition is not installable
  - Any of the Table Descriptor mappings owned by Workflow activities are incomplete
  - There is circular mapping; that is, a Program within the Workflow reads from a Table instance that a subsequently executed Program writes to

Component Object Types

Component objects are contained in other objects. They are not installed directly in a Work Area but only through the objects in which they are contained. If a component object is not installable, its parent object is not installable either.

The following component objects have a status:

- **Notification:** A Notification definition is always installable.
- **Notification Instance:** A Notification instance is also always installable. If no primary timeout period is defined, the system uses zero (0) as the default, and the installation of the Workflow instance in which it is contained does not fail.
- **Parameter:** A Parameter definition is always installable.
- **Parameter Instance:** A Parameter instance is always installable.
- **Parameter Set:** A Parameter Set definition is always installable.
- **Parameter Set Instance:** A Parameter Set instance is always installable.
- **Planned Output:** A Planned Output is always installable.
- **Source Code:** A Source Code definition is not installable if it does not contain a source code file.
- **Source Code Instance:** A Source Code instance is not installable if its Source Code definition is not installable.

- **Table Descriptor:** A Table Descriptor is not installable if its source Table definition is not installable.
- **Variable:** A Variable is always installable.

Organizational Objects

There are only three organizational object types: Domains, Application Areas, and Work Areas. Of these, only Work Areas are ever installed, and only Work Areas have a status.

A Work Area is not installable if:

- It has a status of Retired, Lock for Install, or Lock for Partial Install.
- Its version is not the most current version.
- It has a usage intent of Production and the installation mode is set to Full.
- The same version of the Work Area has already been successfully installed.

> **Note:**
>
> If the same version of the Work Area has already been successfully installed you cannot run an Upgrade installation unless you use Force Regeneration.

- The source definition of one or more object instances included in the installation have been explicitly checked out by a user other than the person initiating the installation.

> **Note:**
>
> It is possible to install a Work Area which has objects checked out by other users if you have the Superuser Checkin Administrator privilege, in addition to normal object security privileges. The system displays a warning indicating that you are checking in objects checked out by other users. It then checks in the objects and installs the Work Area.

B

Object Ownership

In the Oracle Life Sciences Data Hub (Oracle LSH), object definitions and instances are "owned" by their immediate container object. The following diagrams show the Oracle LSH data model for object ownership:

- Figure B-1
- Figure B-2
- Figure B-3

Object ownership has ramifications in many areas of Oracle LSH, including security (user group assignments) and classification; it is possible for objects to inherit the user group assignments and classifications of their owning objects. In addition, Oracle LSH enforces unique names for objects of the same type in the same container object.

- Domains
- Application Areas
- Work Areas

Domains

Domains contain Application Areas and object definitions. Depending on the setting of the Object definitions contained directly in a Domain constitute the Domain library.

In addition, Domains can contain child Domains that themselves contain child Domains, up to nine (9) levels, depending on the value of the Domain Nest Value profile setting for your LSH implementation. A Domain can contain any number of Domains at a single level. For example, if the Domain Nest Value profile is set to one (1), a top-level Domain can contain any number of child Domains, but those child Domains cannot contain any child Domains of their own.

The various objects contained within a Domain are as follows:

- Application Areas. The objects contained in the Application Area are listed separately.
- Table Definitions. These contain—Columns and Constraints.
- Workflow Definitions. These contain:
  - Execution Templates containing Parameter Set Instances that in turn, contain Parameter Instances
  - Load Set Instances
  - Data Mart Instances
  - Report Set Instances
  - Program Instances
  - Parameter Set Instances that contain Parameter Instances

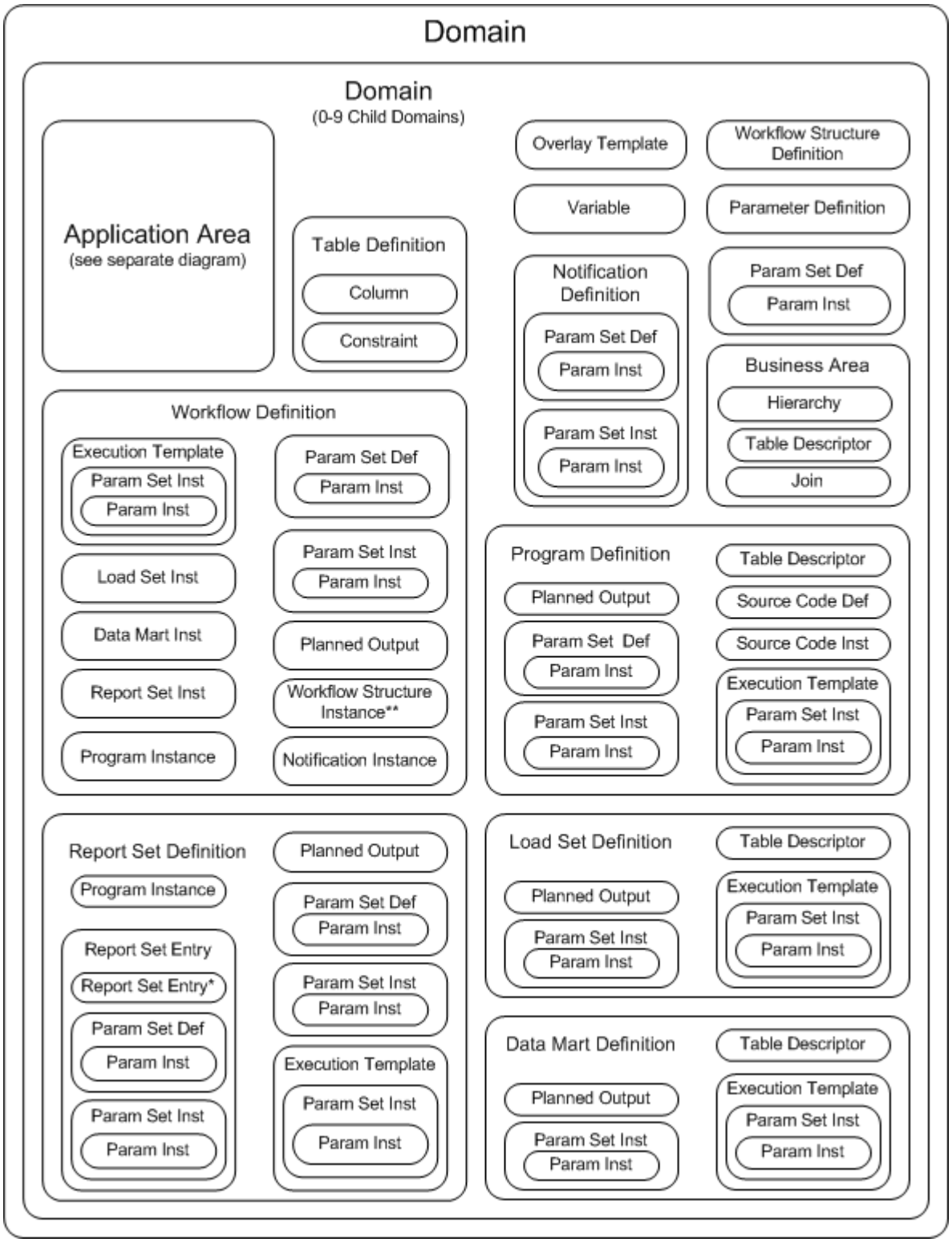
  - Parameter Set Definitions that contain Parameter Instances
  - Planned Outputs
  - Workflow Structure Instances
  - Notification Instances
- Report Set Definitions. These contain:
  - Program Instances
  - Report Set Entries. These contain—Report Set Entries, Parameter Set Definitions containing Parameter Instances, Parameter Set Instances containing Parameter Instances
  - Planned Outputs
  - Parameter Set Instances that contain Parameter Instances
  - Parameter Set Definitions that contain Parameter Instances
  - Execution Templates. These contain Parameter Set Instances that in turn, contain Parameter Instances.
- Overlay Templates
- Variables
- Notification Definitions. These contain Parameter Set Definitions containing Parameter Instances and Parameter Set Instances containing Parameter Instances.
- Workflow Structure Definitions
- Parameter Definitions
- Parameter Set Definitions containing Parameter Instances
- Business Areas. These contain—Hierarchies, Table Descriptors, Joins.
- Program Definitions. These contain:
  - Planned Outputs
  - Parameter Set Definitions that contain Parameter Instances
  - Parameter Set Instances that contain Parameter Instances
  - Table Descriptors
  - Source Code Definitions
  - Source Code Instances
  - Execution Templates. These contain Parameter Set Instances that in turn, contain Parameter Instances.
- Load Set Definitions. These contain:
  - Planned Outputs
  - Parameter Set Definitions that contain Parameter Instances
  - Parameter Set Instances that contain Parameter Instances
  - Table Descriptors
  - Execution Templates. These contain Parameter Set Instances that in turn, contain Parameter Instances.

- Data Mart Definitions. These contain:
  - Planned Outputs
  - Parameter Set Definitions that contain Parameter Instances
  - Parameter Set Instances that contain Parameter Instances
  - Table Descriptors
  - Execution Templates. These contain Parameter Set Instances that in turn, contain Parameter Instances.

Figure B-1 shows all layers of object ownership within a Domain, except for objects contained in the Application Area. See Figure B-2 for object ownership within an Application Area.

> **Note:**
>
> Report Set Entries inside Report Set Definitions can be nested indefinitely.

Figure B-1 Object Ownership within a Domain

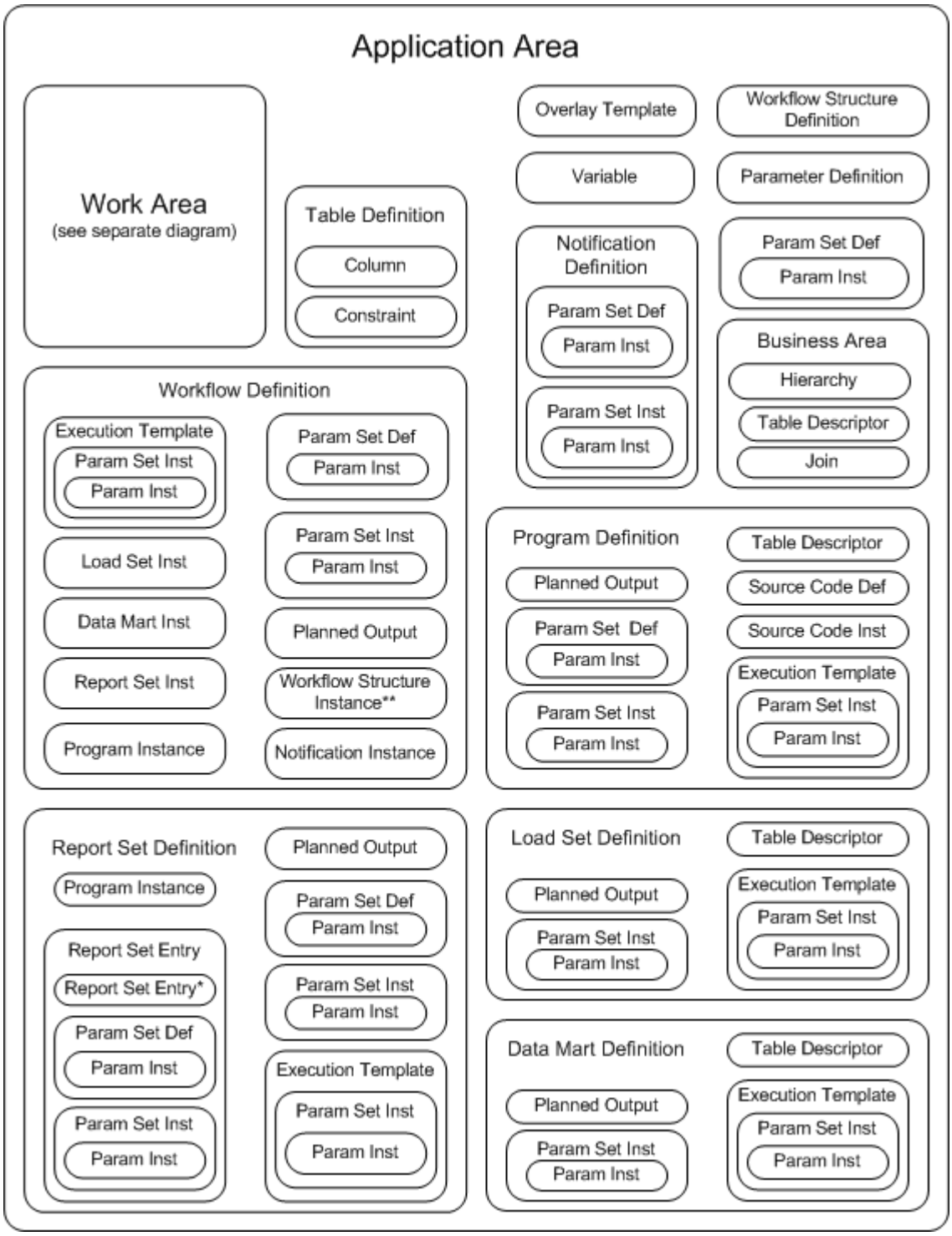


Application Areas

Application Areas contain exactly the same objects as Domains, except that Domains contain Application Areas and Application Areas contain Work Areas. Both Domains and Application Areas can include all object definition types. Figure B-2 shows all possible layers of object ownership within an Application Area.

> **Note:**
>
> Report Set Entries inside Report Set Definitions can be nested indefinitely.

Figure B-2 Object Ownership within an Application Area



Work Areas

Work Areas contain object instances, each of which points to an object definition located either in an Application Area or in a Library. This diagram shows only object

ownership, so you cannot see the pointers to the definitions. Most object instances point to an object definition with the same type name; for example, Program instances point to Program definitions. There are two exceptions: both a Parameter and a Table Column point to a Variable as their definition source.

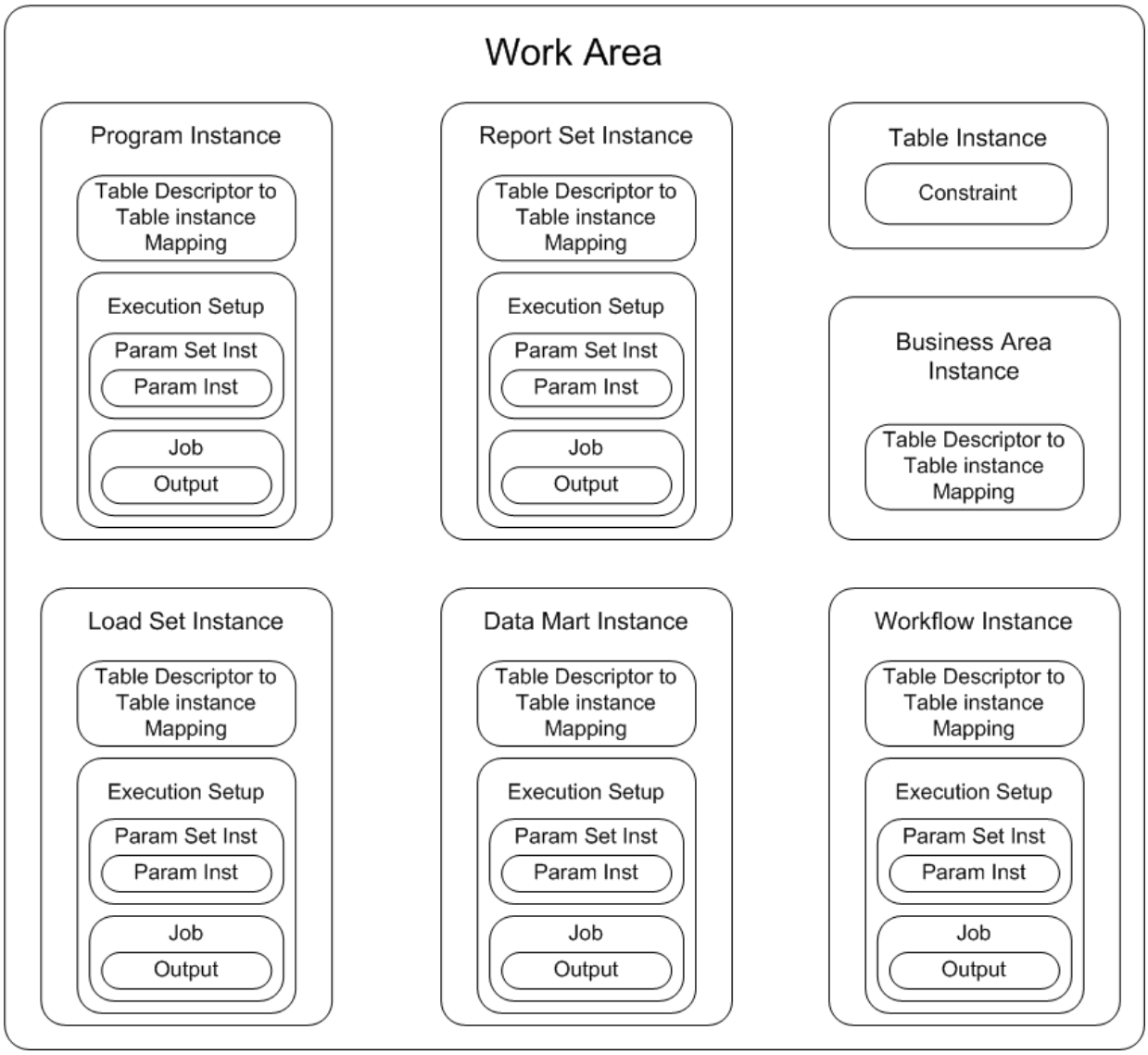
Figure B-3 shows all possible layers of object ownership within a Work Area. The objects are Program Instances, Load Set Instances, Report Set Instances, Data Mart Instances, Work Flow Instances, each of which contain the following objects:

- Table Descriptor to Table Instance mapping
- Execution Setups. These contain Parameter Set Instances containing Parameter Instances, and Jobs containing Outputs.

In addition, Work Areas also contain:

- Table Instances that contain Constraints
- Business Area Instances that contain Table Descriptor to Table Instance mappings

Figure B-3 Object Ownership within a Work Area



Index

A

Actions drop-down list, *2-4*
Administration tab, *3-34*
Advanced Search utility, *3-3*, *3-30*
Appendix B, "Object Ownership", *2-2*
application areas, *B-5*
Application Areas, *2-1*
application life cycle, *12-26*
Applications
    Development screen, *2-13*
    screen, *2-2*
        Create Child screen, *2-8*
    tab, *2-1*
approvals
    using, *10-11*
Automatic Mapping By Name job, *1-9*

B

backchaining, *1-13*, *13-12*
    Execution Setups, *3-71*
    rules, *13-15*
    tips, *13-15*
blind break
    privileges, *4-9*
breadcrumbs, *2-4*
    links, *2-4*
    not working, *2-4*
business areas
    attributes and parameters, *11-12*
    creating, *11-2*
    creating definition and instance, *11-3*
    defining hierarchies, *11-10*
    deploying the RPD file, *11-15*
    generic visualization, *11-18*
    GV BAs
        launching visualizations, *11-21*
    installing instances, *11-22*
    modifying, *11-23*
    OBIEE, *11-12*
    Properties screen, *11-4*
    source code, *11-12*
    visualizing, *11-16*

C

CDISC data model, *1-12*
CDR_TABLESPACE_NAMES lookup, *4-10*
Child Domain
    Select Child drop-down list, *2-7*
classification
    assignment type
        explicit, *3-31*
    details, *3-31*
    hierarchies, *3-31*
    Input/Output Parameters, *3-33*
    planned outputs, *5-27*
columns
    mapping
        columns to constants, *3-58*
        different data types, *3-58*
    names
        special characters, *3-51*, *4-12*
component object types, *A-3*
constraints, *4-14*
container objects, *1-4*

D

data auditing, *13-8*
data blinds
    creating, *13-16*
data currency
    backchaining, *1-14*
    Most Current Available parameter, *1-14*
data marts
    creating, *8-2*
    creating definition and instance, *8-3*
    installing Instances, *8-15*
    modifying, *8-16*
    modifying definition properties, *8-17*
    Oracle Export, *8-13*
    overview, *8-2*
    planned outputs, *8-7*
    Properties screen, *8-4*
    SAS, *8-12*
    setting parameter values, *8-6*
    text, *8-8*
    types, *8-8*

data processing types, *13-2*
data snapshots, *13-9*
data subsets processing, *13-10*
designing an Organizational Structure, *2-4*
displaying
  200 rows, *2-5*
  maximum, *2-5*
domains, *B-1*
Domains, *2-1*
  changing, *2-2*
  child, *2-3*
  Create Application Area screen, *2-7*
  Create Child icon, *2-7*
  creating
    Create Domain screen, *2-6*
    LSH Bootstrap Security application role, *2-5*
    user groups, *2-5*
  default, *2-1*
  Move Objects screen, *2-11*
  nesting
    Domain Nest Value, *2-5*
    Setting the Maximum Number of Nested Domains, *2-5*

E

execution
  overview, *13-1*
  submitting jobs, *13-2*
Execution Setups
  backchaining, *1-14*
  creating, *3-64*
  implicit checkins, *3-77*
  implicit checkouts, *3-77*
  manual checkins, *3-39*
  overview, *3-64*
  synchronization, *3-76*
  undoing checkouts, *3-77*
external system submission, *13-21*

F

FND
  View Object Max Fetch Size, *2-5*
Focus column, *2-3*

I

implicit check out, *1-6*
implicit checkins, *1-6*, *3-77*
installable instances, *A-1*
installation results, *12-18*
installing Load Set Instances, *7-42*

J

joins
  defining, *11-7*

L

Load Set Definitions
  modifying properties, *7-44*
Load Set Instances
  installing, *7-42*
  modifying properties, *7-44*
load sets
  creating, *7-4*
  creating definition and instance, *7-5*
  modifying, *7-43*
  Oracle Clincial Labs, *7-36*
  Oracle Clinical Data Extract Oracle Views, *7-28*
  Oracle Clinical Data Extract SAS Views, *7-27*
  Oracle Clinical Global Library, *7-34*
  Oracle Clinical Randomization, *7-37*
  Oracle Clinical Stable Interface Tables, *7-25*
  Oracle Clinical Study Data, *7-40*
  Oracle Clinical Study Design and Definition, *7-30*
  Oracle Tables and Views, *7-11*
  overview, *7-1*
  planned outputs, *7-10*
  runtime parameters, *7-9*
  SAS, *7-13*
  table descriptors, *7-8*
  text, *7-18*
  types, *7-10*

M

Maintain Domains screen, *2-6*, *2-10*
Manage Definitions button, *2-3*
Manage Security screen, *2-6*, *3-35*
message-triggered submission, *13-21*
Modify privileges, *1-6*
modifying, *9-15*

N

Not Ready icon, *3-78*
notifications
  creating, *10-11*
  defining, *10-10*
  modifying, *10-15*

O

OBIEE
  administration tool, *11-17*
OBIEE answers, *11-16*
OBIEE BAs
  defining, *11-12*
  overview, *11-12*
OBIEE security, *11-17*
Object Definitions and Instances
  upgrading Object Instances, *3-18*
    Upgrade All Instances option, *3-19*
  viewing Object Definitions
    version history, *3-11*
object ownership
  application areas, *B-5*
  domains, *B-1*
  work areas, *B-6*
objects, *A-4*
  installing, *12-18*
  name resolution, *12-21*
Objects, *2-2*
  checking in
    automatic checkin, *3-39*
    Check In screen, *3-16*
    Checkin Administrator privileges, *3-16*
  checking out
    Check Out screen, *3-14*
    Checkin Administrator privileges, *3-14*
  cloning
    Clone Instances screen, *3-26*
    required privileges, *3-24*
  component versions
    version numbers, *3-18*
  containers, *1-4*
  copying
    Copy Objects screen, *3-22*
  definition properties, *1-7*
  Definitions
    checking out, *1-6*
    copying, *2-12*
    creating, *2-13*
    managing, *2-3*
    moving, *2-13*
    names, *2-3*
    revalidating, *1-8*
    standards, *1-12*
  descriptions
    2000 characters limit, *3-7*
  instances, *2-3*
  Maintain Library screen, *2-12*
  moving
    exceptions, *3-27*
    Move Objects screen, *3-27*
  nodes, *2-2*
Objects *(continued)*
  removing
    multiple objects, *3-29*
    object metadata, *3-28*
    rules, *3-28*
  reordering and renumbering
    Renumber flag, *3-42*
    Reorder shuttle, *3-42*
  secondary, *1-5*
  validating
    tools, *3-37*
  versioning, *3-12*
  versions
    checking out, *3-13*
    checkout comment, *3-13*
    old versions, *3-13*
Oracle Business Intelligence Enterprise Edition, *1-4*
Oracle Export data marts, *8-13*
Oracle LSH Context Index Refresh, *3-32*
Oracle LSH Context Index Refresh program, *3-32*
Oracle XML Publisher, *1-4*
organizational objects, *A-4*
Outputs, *3-32*
  viewing
    Planned Output subtab, *3-87*
overlay templates, *9-6*
  creating, *9-7*
  creating definitions, *9-8*
  overview, *9-6*

P

parameters, *6-6*
  allowed values, *6-13*
  creating, *6-9*
  defining details, *6-12*
  getting values from other parameters, *6-20*
  propagating values, *6-17*
  validation rules, *6-14*
Parameters, *1-4*
  sending values to other Parameters, *6-19*
planned outputs
  classification, *5-27*
  defining, *5-26*
  viewing assignments, *9-40*
Planned Outputs
  Planned Output Properties screen, *3-33*
  Planned Outputs tab, *3-33*
plus icon (+), *2-3*
Preferences
  link, *2-1*
  setting, *2-1*

Programs
validation statuses, 3-37
Properties screen, 1-7, 2-4

R

Ready icon, 3-78
reload processing, 13-5
report sets
assigning planned outputs, 9-38
building, 9-15
concurrent editing, 9-4
creating, 9-10
creating definition and instance, 9-11
formatting, 9-6
generation through programs, 9-35
installing, 9-44
modifying, 9-55
narratives, 9-34
overview, 9-2
parameters, creating and setting, 9-21
planned outputs, 9-55
post-processing parameters, 9-24
programs, 9-35
Properties screen, 9-12
report set entries, 9-28
structure view, 9-14
using, 9-4
validating, 9-46
viewing planned outputs, 9-40
Report Sets
narrative, 1-3
reports, 14-1, 14-2
container reports, 14-6
data blinding reports, 14-4
object metadata reports, 14-9
security reports, 14-2

S

SAS data marts, 8-12
Search and Select window, 2-2
Search By drop-down list, 3-5
Search icon, 3-4
Search User Group screen, 2-6
Search utility, 3-3
security reports, 14-2
shared checkout system, 1-6
staging processing, 13-6
supporting documents
Manage Supporting Documents screen, 3-40
status, 3-40
supporting Outputs
Manage Supporting Outputs screen, 3-40
system parameters, 3-71
blind break, 3-73
defining Execution Setups, 3-71
modifying, 3-70
notifications, 3-72
settings, 3-70
submission modes
default, 3-73
system reports
listing, 14-2
running, saving, 14-1

T

table descriptors
defining, 11-6
Table Descriptors, 1-8
defining, 8-6
mapping to Table instances, 3-52
simultaneous creating and mapping with TIs, 3-60
source Table Descriptors
Is Target attribute, 3-45
Upload Table Descriptors screen, 3-46
tables
data blinds, see data blinds, 13-16
new Table instance, creating, 4-4
pass-through views, 13-8
processing types, 13-3
refresh groups, 13-10
reload processing, 13-5
staging processing, 13-6
THT processing, 13-6
transactional processing, 13-4
Tables
attributes, 3-48
Read privileges, 3-45
Table Instances screen, 3-46
text data marts, 8-8
timestamps, 3-82
transactional high throughput processing, 13-6
transactional processing, 13-4
transitions
creating, 10-16
defining, 10-16
overview, 10-16

U

unique keys, 4-8
user groups
revoking assignments, 3-36

V

validation
automatic checkins, *3-39*
security privileges, *3-39*
Validation statuses
Development, *3-12*
Production, *3-5*
Quality Control, *3-17*
Retired, *3-5*
variables
overview, *6-1*
Variables, *1-4*
View Existing Outputs screen, *3-88*
viewing data
IDE, *3-85*
program-generated reports, *3-85*
visualizations, *3-85*
viewing jobs
job columns, *3-86*
Show filter, *3-86*
visualizations
overview, *11-1*

W

WebLogic server
starting, *11-16*
work areas, *B-6*
cloning, *12-25*, *12-29*
installation history, *12-3*
installation processes, *12-20*
installing, *12-13*
personalizing Properties screen, *12-12*
Properties Screen, *12-1*
running, *12-16*
usage intent, *12-5*
version history, *12-4*
Work Areas, *2-1*
cloning, *3-37*
full-mode installations, *4-8*
upgrade-mode installations, *4-8*
usage intent, *3-37*
workflows
adding executables, *10-6*
adding structures, *10-8*
creating, *10-3*
defining parameters, *10-17*
installing instances, *10-18*
mapping table descriptors, *10-8*
modifying, *10-19*
overview, *10-1*
Properties screen, *10-5*
structure types, *10-9*
Workflows
creating definition and instance, *10-4*
notifications, *1-4*
scheduling, *1-14*