Oracle® Life Sciences Data Hub

Adapter Toolkit Guide

Release 2.4.8
E95834-02
January 2020

Oracle Life Sciences Data Hub Adapter Toolkit Guide, Release 2.4.8

E95834-02

Contents

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 About Adapters

This section contains the following topics:

- About Adapters
  Oracle Life Sciences Data Hub is designed for integration with other systems.
- Adapter Components
  An adapter includes a number of components. You should understand these components and their relationships when developing adapters.
- Components and the User Interface
  Your adapter allows Oracle Life Sciences Data Hub Definers to create a new type of Load Set, Program, Data Mart, or Business Area.

About Adapters

Oracle Life Sciences Data Hub is designed for integration with other systems.

You can:

- Load data into Oracle Life Sciences Hub from another system
- Copy Oracle Life Sciences Hub data into a file for export
- Transform and report Oracle Life Sciences Hub data using external technologies as integrated development environments (IDEs)
- Allow read-only access to specified Oracle Life Sciences Hub data through a visualization tool

Oracle Life Sciences Hub ships with adapters to other systems for each of these purposes. If you want to integrate Oracle Life Sciences Hub with a different system, you can create your own adapter.

There is a specialized object type in Oracle Life Sciences Hub for each of these purposes: Load Sets for loading data, Data Marts for exporting data, Programs for manipulating and reporting on data, and Business Areas for specifying the data available to a visualization tool. Oracle Life Sciences Hub developers, or Definers, define an object of the appropriate type in order to accomplish any of these tasks. When you create an adapter, you make it possible for Definers to create a new type of Load Set, Data Mart, Program, or Business Area that uses the logic required by your system.

For example, if you create a Load Set adapter, when a Definer creates a new Load Set in Oracle Life Sciences Hub, your adapter appears in the **Load Set/Adapter Type** drop-down list in the Create Load Set page. If the Definer selects your adapter type, the attributes you have defined for the adapter (if any) appear on the Load Set definition page and the Definer must provide values for them and specify the source data files or tables to complete the Load Set definition. Using the new Load Set, Definers and other users can load data into Oracle Life Sciences Hub from your external system. Your adapter type is also available to the public API for creating Load Sets.

Oracle Life Sciences Hub includes the following adapters:

- **Load Set adapters**. Oracle Life Sciences Hub ships with Load Set adapters for SAS and text files, Oracle tables and views, and Oracle Clinical.
- **Data Mart adapters**. Oracle Life Sciences Hub includes Data Mart adapters to copy Oracle LSH data to SAS data sets, SAS CPort and XPort files, text files, and Oracle Export files.
- **Program adapters**. Oracle Life Sciences Hub includes Program adapters to allow users to write data transformation and/or reporting programs in SAS, SQL Developer, SQL*Plus, Oracle Reports, Oracle Business Intelligence Publisher, and Informatica PowerCenter.
- **Business Area adapters**. Oracle Life Sciences Hub includes Business Area adapters for Oracle Business Intelligence Enterprise Edition (OBIEE), including its Administration tool and Presentation Services.

> **Note:**
>
> Oracle Life Sciences Hub also includes a Generic Visualization Adapter that is available for use with other visualization tools. To use this adapter, follow instructions in Using the Generic Visualization Adapter. You do not need to follow instructions in any other chapter.

Adapter Components

An adapter includes a number of components. You should understand these components and their relationships when developing adapters.

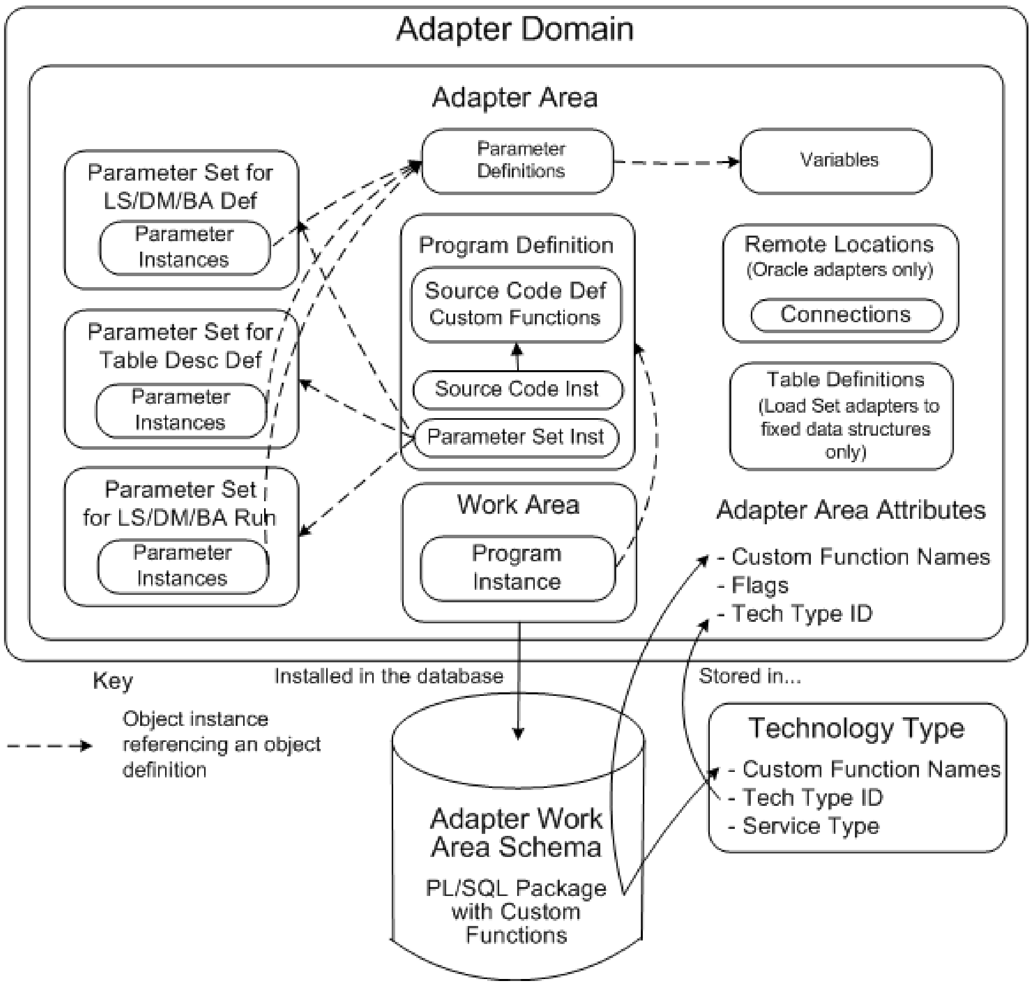
An adapter includes the following components.

See Designing an Adapter for more information.

- **Defined Objects**. You must created defined objects in Oracle Life Sciences Data Hub to create an adapter in the same way you do to create an application in Oracle Life Sciences Hub, in a similar hierarchy; see the *Oracle Life Sciences Data Hub Application Developer's Guide* for more information. You use public APIs to create these objects. The following objects, which are illustrated in Figure 1-1, are required except as noted:
  - **Adapter Domain**. Each adapter must have one Adapter Domain to hold all the other defined objects.
  - **Adapter Area**. Each adapter must have one Adapter Area in the Adapter Domain to hold object definitions and the adapter's Work Area. If your external system interacts with Oracle Life Sciences Hub in multiple ways, you may need to create multiple adapters. You can create a single Adapter Domain containing one Adapter Area for each adapter.
    An Adapter Area has attributes that identify functions you write that are required during Load Set, Data Mart, Program, or Business Area definition, installation, and execution. Other attributes determine which buttons are enabled in the user interface and which functions are required for your adapter. The Adapter Areas table (CDR_ADAPTER_AREAS) also includes a column for the Tech Type ID.

  - **Work Area.** Each Adapter Area must have one and only one Work Area in the Adapter Area to hold an instance of the Program. You *install* the Work Area and Program instance to create a database schema for the adapter that contains the PL/SQL packages that you write.
  - **Program**. Each adapter must have a Program definition inside the Adapter Area and an instance of it in the Work Area.
    If more than one person is writing the functions required for the adapter, you may want to put each person's package in a different Program so that they can install and test their functions at different times.
  - **Source Code**. Each adapter must have at least one Source Code definition and instance inside the Program to contain the PL/SQL functions and procedures you write. You can include all functions and procedures in one or more defined Source Code objects (one PL/SQL package per Source Code object) in one or more Program objects.
  - **Parameters and Parameter Sets**. If your adapter requires user input, you must define Parameters and their underlying Variables to receive the input. Each Parameter you define must be contained in a Parameter Set with a predefined name. The system looks for each Parameter Set by name at a different time during object definition and execution.
  - **Remote Locations and Connections**. If your adapter is for an Oracle-based product, you need to create defined objects to register remote locations and connections.
  - **Tables**. If your system has a fixed data structure from which you want to load data into Oracle Life Sciences Hub, you may want to define that structure in the adapter itself.
- **Technology Type**. You run an API to add a row to the internal Tech Types table (CDR_TECH_TYPES). This table has columns to store the names of functions you write that are required during Load Set, Data Mart, Program, or Business Area installation or execution.
- **PL/SQL Functions and Procedures**. For each action a user performs in the user interface (UI) to define or install a Load Set, Data Mart, Program, or Business Area, or to run a Load Set, Data Mart, or Program or launch a visualization tool, you must write a PL/SQL function or procedure to do the work. You must also enter the name of each function or procedure as a value in the appropriate column when you run the Create Tech Type or Create Adapter Area API.
  You write these functions and procedures in one or more PL/SQL packages and upload each package as the source code file for a defined Source Code object in the defined Program in your Adapter Area.
- **Service Type(s)**. Create a new service type for each new technology type.
- **Lookup Values**. Extend the Technology Types and Service Types lookups for each technology type and service type you add. If you need new file types, extend the File Types lookup as well.

Figure 1-1   Adapter Components and Their Relationships

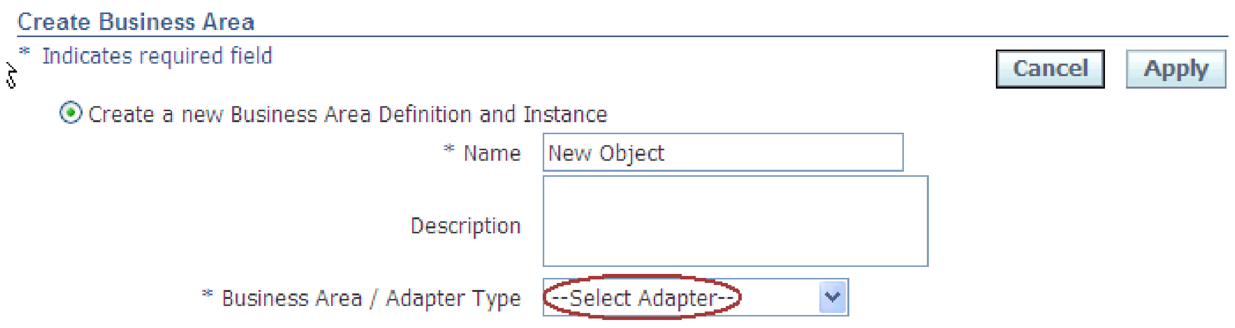


The diagram shows the metadata object owning relationships for the adapter. All metadata is contained in the Adapter Domain except the technology type. The arrows in the Adapter Domain portion of the diagram show references between object definitions and instances; these are standard Oracle Life Sciences Hub object relations and you create the definition and instance at the same time, using an API. The Work Area is installed as a set of database schemas that store the custom functions and procedures you write. Each custom function's name is stored in a column of either the Adapter Areas or Tech Types table (CDR_ADAPTER_AREAS or CDR_TECH_TYPES). Each Adapter Area stores the ID of its primary technology type.

Components and the User Interface

Your adapter allows Oracle Life Sciences Data Hub Definers to create a new type of Load Set, Program, Data Mart, or Business Area.

Figure 1-2 Selecting the Adapter Type During Object Creation



Create Business Area

* Indicates required field

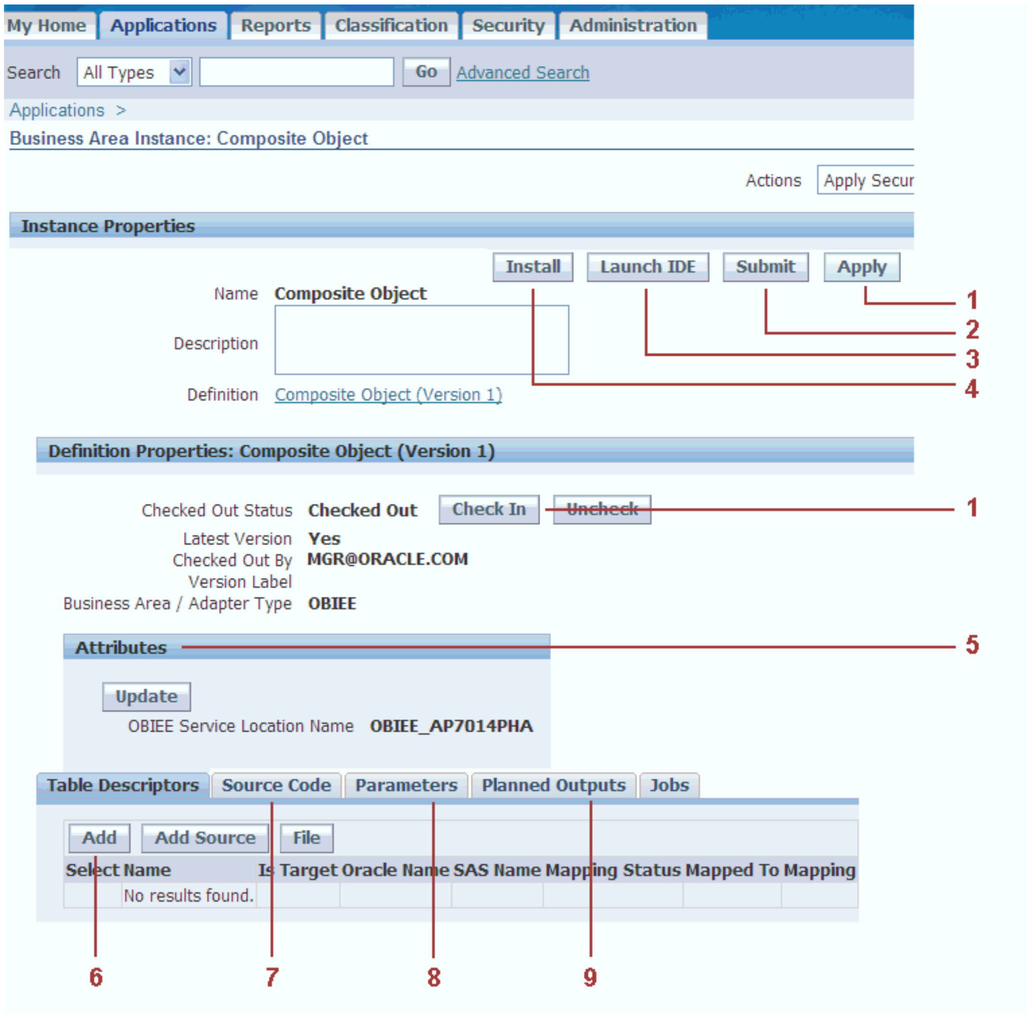
Cancel Apply

● Create a new Business Area Definition and Instance

* Name	New Object
Description	
* Business Area / Adapter Type	--Select Adapter--

When a Definer creates an object of any type, he or she must select an adapter type. The system adds the name you define for the Adapter Area to the list.

Figure 1-3   Object Properties User Interface



Business Area Instance: Composite Object

Name: Composite Object
Description:
Definition: Composite Object (Version 1)

Checked Out Status: Checked Out
Latest Version: Yes
Checked Out By: MGR@ORACLE.COM
Version Label:
Business Area / Adapter Type: OBIEE

OBIEE Service Location Name: OBIEE_AP7014PHA

No results found.

Note:

The above "screenshot" is an amalgam of buttons and options from different object types and states.

User interface elements for all objects interact as follows with adapter components:

1. The **Apply** button calls the status_recalc_function for Data Marts. For Programs and Load Sets this function is called on checkin and checkout; see Status_Recalc_Function .
2. The **Submit** button (available for Programs, Load Sets, and Data Marts only) opens the Execution Setup page displaying the Parameters in the runtime Parameter Set PARAMETERSET_LOADSETLEVEL_RUN, if any, so that the user can enter values and click **Submit** again to actually run the Program, Load Set, or Data Mart. The system then automatically generates and dequeues an XML message file and calls the execution function(s) specified for the technology type and Adapter Area. Execution also requires a service type and an execution

command to invoke the external system; see Planning for Object Execution and Object Execution Functions and Procedures.

3. The **Launch** button (available for Programs and Business Areas only) invokes the custom function Build_IDE_Cfg_Function. This button becomes active after the user installs the current version of the Program or Business Area.

> **Note:**
>
> Launch IDE functionality is not available using the Generic Visualization adapter.

4. The **Install** button calls the installation function(s) specified for the technology type; see Object Installation Functions .
5. Object **Attributes** are the Parameters you define in the Parameter Sets named PARAMETERSET_LOADSETLEVEL_DEF and PARAMETERSET_OPERATORLEVEL. They collect user input to use during object definition and Table Descriptor definition respectively; see Planning Parameters and Parameter Sets.
6. The **Add** Table Descriptor button requires the Adapter Area flag allow_manual_tab_desc_flag to be set to **Yes** to be active. Users can then manually add Table Descriptors. If this flag is set to **No**, the button is not available and the adapter define-time functions must create the Table Descriptors required.

   The **Upload** and **Upload Columns** buttons for Programs and Load Sets are displayed if the AllowAutoAddTabDesc and Allow_column_upload flags, respectively, are set to **Yes** or **File**. You can then write custom definition procedures that call APIs to upload either the whole Table Descriptor structure or just the Columns.

   See Object Definition Functions and Procedures for information on the related functions.
7. The **Source Code** tab is available for Programs and Business Areas only. No adapter-specific code is required to upload a source code file from the integrated development environment (IDE).
8. The **Parameters** tab can contain Parameters that you define in the adapter Parameter Set named PARAMETERSET_LOADSETLEVEL_RUN to collect user input. For example, all SAS Data Marts have predefined Parameters for Mode (CPORT or XPORT) and Zip Result Flag (No or Zip) that the Definer can set during Data Mart definition or the user can set at runtime.

   The Parameters tab can also contain Parameters created by a Definer. These are contained in a different Parameter Set and are handled automatically by the system; you do not need to write code or add parameters to any adapter Parameter Set to support this.

   You may need to write your execution function to handle any additional Parameters the definer may create and pass them to the particular program at execution.
9. **Planned Outputs** You can use the define_time_function to create predefined Planned Outputs for all objects created using your adapter; for example, as a placeholder for Data Mart data files or log or error files for any object type.

In addition, you can use the status_recalc_function, which is invoked each time an object is modified, to automatically create a Planned Output when a Definer adds an object that requires it; for example:

- Write the status_recalc_function so that it automatically creates a new data file for the new Table Descriptor each time a Definer adds a Table Descriptor to a Data Mart of your adapter type.
- Write the status_recalc_function so that it automatically creates a new log and/or error file each time a Definer adds a Source Code object to a Program.

The system automatically provides the functionality for Definers to create Planned Outputs in Program definitions.

2 Designing an Adapter

This section contains the following topics:

- Preparation
  Before attempting to design your adapter, you can prepare by reading related topics and gathering important information.
- Requirements for Load Set, Data Mart, Program, and Business Area Adapters
  The functionality required for an adapter depends in large part on whether the adapter's purpose is to load data intoOracle Life Sciences Data Hub (Load Set adapters), export data from Oracle Life Sciences Hub (Data Mart adapters), transform data (Program adapters), or to allow an external tool to view data contained in Oracle Life Sciences Hub(Business Area adapters). However, the requirements also vary depending on the external system.
- Planning Adapter Areas
  Most external systems require only a single Adapter Area but, depending on your situation, you may need to implement multiple adapter areas.
- Planning Technology Types
  You must create at least one new technology type for each Adapter Area.
- Planning Services
  Services are required to handle the interaction of Oracle Life Sciences Data Hub with the external system.
- Planning PL/SQL Functions and Procedures
  For each action a user performs in the user interface (UI) to define or install a Load Set, Data Mart, Program, or Business Area, or to run a Load Set, Data Mart, or Program or launch an IDE, you must write a PL/SQL function or procedure to do the work.
- Planning Parameters and Parameter Sets
  You must plan parameters and parameter sets to get information from the user.
- Adding Lookup Values
  You must extend certain lookups. You can also enable your customers to add lookup values in their own environment.
- Planning Planned Outputs
  Data Mart adapters may require predefined Planned Outputs to support the actual data file produced by running the Data Mart.
- Planning Data Structures
  If you are creating a Load Set adapter and your source data system has fixed data structures, you can define Tables in your Adapter Area with the same structure.
- Planning for Object Execution
  Programs, Load Sets, and Data Marts are all executable objects and an adapter of these types must handle the object execution. Business Area and Program IDE adapters may also require execution functionality.

- Planning Security
  The security required for your adapter depends on the interaction with the external system that it requires.
- Planning Integrated Development Environment Adapters
  There are a number of things your Definers may need to be enabled to do in an integrated development environment (IDE). You must consider these needs when planning IDE adapters.

Preparation

Before attempting to design your adapter, you can prepare by reading related topics and gathering important information.

To prepare for designing an adapter, you may find it helpful to:

- Read About Adapters and Developing an Adapter.
- Familiarize yourself with the Oracle Life Sciences Data Hub user interface for creating the relevant object type—Load Set, Data Mart, Program, or Business Area—and read the corresponding chapter of the *Oracle Life Sciences Data Hub Application Developer's Guide*.
- You may want to create a Load Set, Data Mart, Program, or Business Area of each existing type in the user interface and compare each one to the Adapter Areas and Tech Types column values for the corresponding adapter type. You can see the actual settings for shipped Oracle LSH adapters through two views, CDR_ADAPTER_AREAS_V and CDR_TECH_TYPES_V.
- Read the chapter on execution in the *Oracle Life Sciences Data Hub Application Developer's Guide* and the chapter on services in the *Oracle Life Sciences Data Hub System Administrator's Guide*. The *Oracle Life Sciences Data Hub System Administrator's Guide* also has chapters on setting up security for some shipped adapters that show different ways of synchronizing security between Oracle Life Sciences Hub and external systems.
- Read the *Reference Information* section about APIs in the *Oracle Life Sciences Data Hub Application Programming Interface Guide*.
- Understand the external system for which you are creating an adapter.

Requirements for Load Set, Data Mart, Program, and Business Area Adapters

The functionality required for an adapter depends in large part on whether the adapter's purpose is to load data intoOracle Life Sciences Data Hub (Load Set adapters), export data from Oracle Life Sciences Hub (Data Mart adapters), transform data (Program adapters), or to allow an external tool to view data contained in Oracle Life Sciences Hub(Business Area adapters). However, the requirements also vary depending on the external system.

- Load Set Adapter Requirements
  Load Set adapters ensure that Load Sets of their type have target Table Descriptors with the same metadata structure as in the source system and must actually transfer data to Oracle Life Sciences Data Hub.

- Data Mart Adapter Requirements
  Data Mart adapters enable users to create data files for their technology type, containing data from one or more Oracle Life Sciences Data Hub Table instances —either all data, with audit information, or all data at a particular timestamp (including current data).
- Program Adapter Requirements
  Program adapters must enable users to develop and run programs.
- Business Area Adapter Requirements
  Business Area, or visualization, adapters enable users to create a Business Area containing Table Descriptors, Joins, and Hierarchies, instantiate them in the Oracle Life Sciences Hub database, and send the metadata to the visualization system in a form comprehensible to the visualization system.

Load Set Adapter Requirements

Load Set adapters ensure that Load Sets of their type have target Table Descriptors with the same metadata structure as in the source system and must actually transfer data to Oracle Life Sciences Data Hub.

Or, if the external system is Oracle-based, you may also be able to provide the option to create pass-through views so that users can view data live in the source system from Oracle Life Sciences Hub.

For more information, see the following:

- Definition
- Installation
- Execution

Definition

During Load Set definition, a Load Set adapter may need to do the following:

1. Receive information from the user on the location from which to upload or copy metadata structures; see Planning Parameters and Parameter Sets.
2. Connect to the source data system.
3. Upload columns or otherwise copy the metadata structure of the data to be loaded to one or more Oracle LSH Table Descriptors and their source Table definitions.

   For some adapters, such as the shipped Text Load Set adapter, this is not possible and the user must manually define the metadata structure (Table Descriptor).

Installation

Most Load Set adapters are not required during Load Set installation.

However, the tech types table stores installation function names and you can write them if you need them; see Object Installation Functions .

Execution

At runtime, a Load Set adapter may need to do the following:

1. Verify that the transport mechanism is available.
2. Verify that the structure of the incoming data is the structure expected by Oracle Life Sciences Data Hub, and return an error if it is not.
3. Verify the source data currency.
4. Write data to targets in Oracle Life Sciences Hub.
5. Return results and status.

Data Mart Adapter Requirements

Data Mart adapters enable users to create data files for their technology type, containing data from one or more Oracle Life Sciences Data Hub Table instances—either all data, with audit information, or all data at a particular timestamp (including current data).

Depending on the technology, it may be appropriate to create one file per Table instance or to combine all data in a single file. It may be appropriate to zip the file.

For more information, see the following:

- Definition
- Installation
- Execution

Definition

During Load Set definition, a Load Set adapter may need to do the following:

1. Receive information from the user on the location from which to upload or copy metadata structures; see Planning Parameters and Parameter Sets.
2. Connect to the source data system.
3. Upload columns or otherwise copy the metadata structure of the data to be loaded to one or more Oracle LSH Table Descriptors and their source Table definitions.
4. For some adapters, such as the shipped Text Load Set adapter, this is not possible and the user must manually define the metadata structure (Table Descriptor).

Installation

Most Data Mart adapters are not required during Data Mart installation.

However, the tech types table stores installation function names and you can write them if you need them; see Object Installation Functions .

Execution

When a user runs a Data Mart, a Data Mart adapter may need to do the following:

1. Verify that the required technology is available.
2. Write Oracle Life Sciences Data Hub data from the Table instances mapped to the Data Mart Table Descriptors to files in the appropriate format.

3. Store the files in Oracle Life Sciences Hub.

Program Adapter Requirements

Program adapters must enable users to develop and run programs.

In detail, a program adapter must do the following:

- Enable users to launch an integrated development environment (IDE) from the Oracle Life Sciences Data Hub Program UI to develop a program that reads data in Oracle Life Sciences Hub tables and either generates a report or manipulates data and writes to target tables.
- Run those user-defined programs in the appropriate processing engine.

The IDE part of the adapter may not require an Adapter Domain or Adapter Area or any other defined objects, but only a technology type and custom functions; see Planning Adapter Areas and Planning Integrated Development Environment Adapters.

When a Definer clicks the Launch button in an installed Program of the technology type you create, the adapter must open the appropriate IDE and make the data in the Table instances mapped to the Program's Table Descriptors available in the IDE.

Since Program IDE adapters allow users to view Oracle Life Sciences Hub data through an external tool, those adapters must coordinate user privileges between Oracle Life Sciences Hub and the external system.

For more information, see the following:

- Definition
- Installation
- Execution

Definition

A Program adapter does not require PL/SQL programs to set up source data structures during Program definition because the source data structures are within Oracle Life Sciences Data Hub.

Programs of the new technology type may require special attributes that you must set up as define-time parameters. Your define_time_function must then collect the user-specified values of these attributes and handle them appropriately.

You may want to call a public API trom the define_time_function to create a Planned Output as a placeholder for the log file; see Planning Planned Outputs.

Write a function to recalculate their Installable status after each Definer modification and checkin; see Status_Recalc_Function.

Installation

You can use installation functions to synchronize Oracle Life Sciences Data Hub object metadata with your adapter's external system.

For example, download the Program's source code or the values of the Program's define-time Parameters; see Object Installation Functions .

Execution

When a user runs a Program, a Program adapter may need to do the following:

1. Verify that the required technology is available.
2. Read Oracle Life Sciences Data Hub data from the Table instances mapped to the Program's source Table Descriptors and execute the user-defined source code in the appropriate processing engine, either to write data to the Table instances mapped to the Program's target Table Descriptors or to generate a report.
3. Check the source data currency.
4. Generate a log file.
5. Store the output(s) in Oracle Life Sciences Hub.

Business Area Adapter Requirements

Business Area, or visualization, adapters enable users to create a Business Area containing Table Descriptors, Joins, and Hierarchies, instantiate them in the Oracle Life Sciences Hub database, and send the metadata to the visualization system in a form comprehensible to the visualization system.

Note:

A Generic Visualization adapter is available to integrate any visualization tool with Oracle Life Sciences Data Hub with much less work than creating your own adapter; see Using the Generic Visualization Adapter.
However, you may want to create your own adapter in order to use the following Business Area functionality, which is not available when you use the Generic Visualization adapter:

- Launch IDE from the Business Area
- Define joins and hierarchies in the Business Area

With support from the adapter, a user can launch the visualization system from Oracle Life Sciences Hub or log in directly to the external system and view data in the Table instances mapped to the Business Area's Table descriptors. The data remains in Oracle Life Sciences Hub.

If further definition is required in the external system, the adapter must facilitate that. You may need to save these externally made changes back to Oracle Life Sciences Hub under version control. The OBIEE adapter uploads a source code file for this purpose.

Since Business Area adapters allow users to view Oracle Life Sciences Hub data through an external tool, those adapters must coordinate user privileges between Oracle Life Sciences Hub and the external system.

For more information, see the following:

- Definition
- Installation

- Launching the Visualization Tool

Definition

A Business Area adapter does not require custom functions to set up source data structures during Business Area definition because the source data structures are within Oracle Life Sciences Data Hub.

However, you may need to use the definition functions for define-time parameter values or source code.

> **Note:**
>
> Joins and hierarchies are automatically available for definition in any Business Area. The adapter does not have to handle this.

Installation

At installation, a Business Area adapter may need to push Oracle Life Sciences Data Hub metadata to the external system and create a corresponding metadata representation appropriate to the visualization system.

At installation, a Business Area adapter may need to push Oracle LSH metadata to the external system and create a corresponding metadata representation appropriate to the visualization system.

For example, the OBIEE adapter creates an OBIEE Subject Area corresponding to each OBIEE Business Area. The first time an OBIEE Business Area is installed, the OBIEE adapter creates a default repository (RPD) file and deploys it on the OBIEE Presentation Server.

Launching the Visualization Tool

You need to provide the URL required to launch the visualization tool and write a function (see Build_IDE_Cfg_Function) that is tracked in the Tech Types table.

For some external visualization systems (for example, OBIEE), you may need to access the external system in one mode from the Business Area UI and in another mode from the Visualizations subtab of the Reports tab. Oracle Life Sciences Data Hub calls the build_ide_cfg_function in both cases, but detects whether the user is in the Business Area or in the Reports tab and calls the appropriate environment.

Planning Adapter Areas

Most external systems require only a single Adapter Area but, depending on your situation, you may need to implement multiple adapter areas.

However, you may want to create multiple Adapter Areas for the same system if, for example, you want to load many different types of data and/or metadata from the same system. This is true of the Oracle Clinical adapter, which has a single Adapter Domain containing multiple Adapter Areas, one for each type of data or metadata loaded.

You need multiple Adapter Areas if the Load Sets, Programs, Data Marts, or Business Areas created for your external system require:

- Different define-time or runtime Parameters; see Planning Parameters and Parameter Sets.
- Different predefined fixed data structures matching those in the external system; see Planning Data Structures.

Your adapter may need multiple technology types but only one adapter type; for example, if your external system is required both as an integrated development environment (IDE) and for execution. The IDE part of the adapter may not require an Adapter Domain or Adapter Area or any other defined objects, but only a technology type and custom functions (primarily the Build_IDE_Cfg_Function). This is true if the IDE requires the user to log in (as SAS does). If the adapter does not require the user to log in , you need to create an Adapter Domain, Adapter Area and other objects.

Planning Technology Types

You must create at least one new technology type for each Adapter Area.

Oracle Life Sciences Data Hub requires that you follow a naming convention to distinguish between technology types:

- If the adapter needs to open an integrated development environment (IDE) you must create a technology type with a name that ends with the string `DEV`. The system calls the Build_IDE_Config_function only for technology types whose name ends with `DEV`.
  This technology type can handle opening two different IDEs if required, using the value that the system supplies to an input parameter of the Build_IDE_Cfg_Function to determine whether the user launched the IDE from the Reports tab or from a Program or Business Area.
- If the adapter does not open an IDE, or if it needs to do something in addition to opening an IDE, create a technology type (or an additional technology type) that does not have a name that ends with `DEV`.

You may need more than one technology type of the non-IDE type if your adapter needs to call two different functions at the same point in processing—that is, if a fork is required.

For example, the Oracle LSH OBIEE adapter has three technology types, CDR$OBIEEDEV for the IDE and two others, CDR$OBIEE and CDR$OBIEETMP. CDR$OBIEE is the base technology type for the OBIEE adapter; the one whose name is in the tech_types column of the adapter area table. Each of the two has both an Execution_Function and a Post_Execution_Function so they cannot coexist in the same technology type. Instead, the Install_Function of the base OBIEE technology type, which the system calls during Business Area installation, calls both execution functions. The execution function of the second technology type is coded to wait until the post-execution function of the first has completed so that the following occur in order:

1. Generate the RPD file.
2. Upload the RPD file to the Business Area Source Code definition.
3. Deploy the RPD file to the OBIEE Presentation Server.
4. Restart the OBIEE Presentation Server.

In addition, if your external system uses multiple types of processing—for example, Java and C++—you must create a technology type for each processing type.

Each technology type definition includes:

- A unique ID. Contact Oracle Support to get an ID for your technology type that is unique across all adapters developed for use with Oracle Life Sciences Data Hub.
- The object type to be created using the adapter.
- A service type; see Planning Services.
- The names of functions and procedures to be used during object installation and execution.
- Oracle Life Sciences Hub uses Oracle Warehouse Builder (OWB) for job tracking. Specify one of two possible OWB operators for adapters that run jobs:
  - **CdrPLSQLImmediate_1** for external systems that use PL/SQL to execute their Load Sets, Programs, or Data Marts.
  - **CdrSERVICE_1** for external systems that are located on an operating system; these adapters use the Oracle LSH Distributed Processing (DP) Server to execute their defined objects.

  No OWB operator is required to launch an IDE, so an IDE technology type may not need an OWB operator. However, if the adapter needs to take an action in the IDE during or after the launch, its technology type needs the **CdrSERVICE_1** OWB operator. For example, the OBIEE 10g adapter needs to copy the RPD file to OBIEE and restart the Presentation Server the first time an OBIEE Business Area is installed. It uses this OWB operator for that purpose.

> **Note:**
>
> There is one additional OWB operator, CdrDiscovererBA, which was hardcoded for the former Oracle Discoverer adapter. Do not use this operator.

Planning Services

Services are required to handle the interaction of Oracle Life Sciences Data Hub with the external system.

You must create at least one service type for your adapter. A different service type is required for each technology type. To create a new service type, extend the CDR_SERVICE_TYPEs lookup; see Adding Lookup Values.

Each Oracle Life Sciences Hub installation that uses your adapter must have a service, service instances, and a service location for each service type required for your adapter defined in the Oracle LSH user interface. Your custom functions can then make use of the local information provided in the service, service location, and service instance definitions, which are stored in the following internal tables:

- CDR_SERVICE_LOCATIONS
- CDR_SERVICES
- CDR_SERVICE_INSTANCES

The system assigns a service instance to each job execution and IDE launch. For information on the function of service instances during job execution, see Execution Process.

If you need to collect additional information about the local installation of Oracle Life Sciences Hub or your external system, you can use the Details field of the service definition to collect it. The Details field is not required. The shipped adapters use it in different ways that are described in the *Oracle Life Sciences Data Hub System Administrator's Guide* chapter on services. The user-entered value of the Details field is stored in the DETAILS column of the CDR_SERVICES table (varchar2(2000) BYTE).

Planning PL/SQL Functions and Procedures

For each action a user performs in the user interface (UI) to define or install a Load Set, Data Mart, Program, or Business Area, or to run a Load Set, Data Mart, or Program or launch an IDE, you must write a PL/SQL function or procedure to do the work.

Oracle Life Sciences Data Hub designers have tried to anticipate every possible type of PL/SQL function or procedure that might be required by any adapter, and have added columns to the Adapter Areas table and the Tech Types table to store their names and call them at particular times for particular purposes. Each function and procedure has a required signature that is described in the following sections.

Note:

You are not required to supply every function and procedure. Write only those that are required for your adapter.

When you run the APIs to create an Adapter Area or Technology Type you enter the name of each function or procedure as the value for the the appropriate attribute.

Oracle Life Sciences Hub calls each function at the appropriate time (object definition, installation, or execution) if its name is in the Adapter Areas or Tech Types table, and automatically passes the required input parameter values to the function.

You include your functions and procedures in one or more PL/SQL packages and upload each package to a Source Code definition that you create inside the Program definition in the Adapter Area. If multiple people are coding your adapter at the same time, you may want to develop these functions and procedures in separate packages; see Planning Programs and Packages.

If you need user input for any of these functions, you need to define Parameter objects to collect the information in the user interface and refer to these Parameters in your source code. See Planning Parameters and Parameter Sets for further information.

For more information, see the following topics:

- Object Definition Functions and Procedures
  All object definition function and procedure names are stored in the adapter_areas table.

- Object Installation Functions
  Depending on the external system, you may need to write functions for use when a user installs a Load Set, Data Mart, Program, or Business Area. All object installation function names are stored in the tech_types table.
- Object Execution Functions and Procedures
  Execution-related functions are stored in either the tech_types table or the adapter_areas table.

Object Definition Functions and Procedures

All object definition function and procedure names are stored in the adapter_areas table.

Note:

Write define-time functions and procedures so that they use UI error handling to display any errors they may return in the user interface.

See details for the following functions:

- Column_Upload_Function
- Auto_Add_Tab_Desc_Function
- Auto_Add_Tab_Desc_LOV
- Define_Time_Function
- Status_Recalc_Function

Column_Upload_Function

(Applies only to Load Set and Program adapters.) Use this function to enable Definers to upload files (such as a SAS data set or XML file) to create Table Descriptors or Table Descriptor columns with the same metadata structure as the file.

You can also use this function to read table metadata in a database and create a Table Descriptor from it.

If you set the Adapter Area flag ALLOW_COLUMN_UPLOAD to **File** (for a file system) or **Yes** (for a database), the Upload Columns button is enabled in the user interface and the system invokes the column upload function when required; see Creating an Adapter Area.

The system creates a BLOB from the file and passes a name/value pair in the parameter pi_cNameValuePair with name = `TMP_BLOB_ID` and a value equal to the BLOB_ID of the uploaded file. You write the column upload function to read the BLOB using the ID passed and process it to create either a Table Descriptor, including its columns, or just columns for an existing Table Descriptor.

Oracle Life Sciences Data Hub has an Upload button for new Table Descriptors on the Table Descriptor subtab on the Load Set and Program definition and instance pages. It also has an Upload Columns button on the Table Descriptor page. The system invokes the Column Upload function when the user clicks the Upload button in either location. You can write the function so that it calls the public API

CDR_PUB_DF_TABLE.UPLOADOPERATORCOLUMNS to upload columns or an entirely new Table Descriptor:

- If you call the API to create an entirely new Table Descriptor, leave the Object ID and version parameters null since they do not exist. The API creates a new Table Descriptor and assigns an Object ID and version number to it.
- If you call the API from the context of an existing Table Descriptor to update its columns, pass the Table Descriptor's Object ID and version number to the API.

In the case of uploading columns to an existing Table Descriptor, the API updates the existing column definitions inOracle Life Sciences Hub if there are differences.

If the adapter uploads database table structures, use the following two definition functions (Auto_Add_Tab_Desc_Function and Auto_Add_Tab_Desc_LOV.

Create Parameters in a Parameter Set with the name `PARAMETERSET_LOADSETLEVEL_DEF` to collect the name and location of the file—or the remote location and connection—from the Definer; see Planning Parameters and Parameter Sets and reference these parameters in your column upload function.

The Column_Upload_Function must have the following signature:

```
PROCEDURE UploadOperatorColumns (pi_nCompanyID IN CDR_DF_NAMING_V.COMPANY_ID%TYPE
  ,pi_nLSRObjID         IN CDR_DF_NAMING_V.OBJ_ID%TYPE
  ,pi_nLSRObjVer        IN CDR_DF_NAMING_V.OBJ_VER%TYPE
  ,pi_nOperObjID        IN CDR_DF_NAMING_V.OBJ_ID%TYPE
  ,pi_nOperObjVer       IN CDR_DF_NAMING_V.OBJ_VER%TYPE
  ,pi_cNameValuePair    IN CDR_NAME_VALUE_PAIR_COLL DEFAULT
NULL) IS
```

The procedure's parameters take values as follows:

- **pi_nCompanyID** see Getting Your Company ID
- **pi_nLSRObjID** takes the Object ID of the Load Set or Program definition.
- **pi_nLSRObjVer** takes the object version of the Load Set or Program definition.
- **pi_nOperObjID** takes the Object ID of the Table Descriptor.
- **pi_nOperObjVer** takes the object version of the Table Descriptor.
- **pi_cNameValuePair** The system creates a BLOB from the uploaded file and passes a name value pair with **name** = `TMP_BLOB_ID` and **value** = *BLOB_ID_of_the_uploaded_file*.

Auto_Add_Tab_Desc_Function

(Applies only to Load Set adapters.) You can write a procedure to retrieve a list of all the tables or other data structures in the external system in a user-specified location and to insert them into a list values so that the user can select some or all of them to be uploaded. The system then calls the auto_add_tab_desc_lov function and passes the user's selection to it.

Note:

The above description is correct. The intended functions for this column in the Adapter Areas table and the Auto_Add_Tab_Desc_LOV column are reversed.

If you write a procedure for this purpose, set the Allow_Auto_Add_Tab_Desc flag to **YES**. If not, set Allow_Auto_Add_Tab_Desc to **NO**. **YES** or **NO** must be in uppercase. If **YES**, you must also write a procedure to create Table Descriptors from the selected source data structures (see following procedure).

The following Oracle Clinical Load Set shipped adapters use an Auto_Add_Table_Descriptor_Function: Data Extract Oracle Views, Data Extract SAS Views, Study Data and Study Design. The shipped Oracle Tables and Views Load Set adapter also uses this function.

The Auto_Add_Tab_Desc_Function must have the following signature:

```
PROCEDURE getDataOperList(pi_nCompanyID   IN CDR_DF_NAMING_V.COMPANY_ID%TYPE
  ,pi_nLSRObjID         IN CDR_DF_NAMING_V.OBJ_ID%TYPE
  ,pi_nLSRObjVer        IN CDR_DF_NAMING_V.OBJ_VER%TYPE
  ,po_vOperList         OUT NOCOPY CDR_VAR_LIST_COLL) IS
```

The function's parameters take the following values:

- **pi_nCompanyID** see Getting Your Company ID
- **pi_nLSRObjID** takes the Object ID of the Load Set definition
- **pi_nLSRObjVer** takes the object version of the Load Set definition
- **po_vOperList** outputs a list of Tables from which the user can choose one or more to upload. The return value should be a collection and the ORACLE_NAME attribute of each CDR_VAR_OBJ_TYPE that is part of the returned collection should be populated with the name of a table.

Auto_Add_Tab_Desc_LOV

(Applies only to Load Set adapters.) if you set the Allow_Auto_Add_Tab_Desc flag to **YES** you must also write a procedure to create Table Descriptors in Oracle LSH based on each of the data structures the Definer selects from the list of values (LOV).

Note:

The above description is correct. The intended functions of this PL/SQL function and the Auto_Add_Tab_Desc_Function PL/SQL function are reversed.

The Auto_Add_Tab_Desc_LOV must have the following signature:

```
PROCEDURE CreateMultipleOperators (pi_nCompanyID  IN CDR_DF_NAMING_V.COMPANY_ID%TYPE
  ,pi_nLSRObjID         IN CDR_DF_NAMING_V.OBJ_ID%TYPE
  ,pi_nLSRObjVer        IN CDR_DF_NAMING_V.OBJ_VER%TYPE
  ,pi_vOperList         IN VARCHAR2) IS
```

The function's parameters take the following values:

- **pi_nCompanyID** see Getting Your Company ID
- **pi_nLSRObjID** takes the Object ID of the Load Set definition
- **pi_nLSRObjVer** takes the object version of the Load Set definition
- **pi_vOperList** takes the list of Tables the user selected to upload. The table names are separated by the pipe character (|) with an additional pipe at the end. For example, if the user selects tables named demog, ae, and conmed, the input value is `demog|ae|conmed|`.

Define_Time_Function

The system launches the Define-Time function from the Create page of a Load Set, Program, Data Mart, or Business Area when the user clicks **Apply**.

If all objects created using your adapter need the same Planned Output, or one or more Table Descriptors with a fixed structure, you can use the Define-Time function to create them automatically for every object of your adapter's type.

For example:

- Add a Planned Output as a placeholder for the log file or Data Mart file; call the public API CDR_PUB_DF_PLANNED_OUTPUT.CREATEPLANNEDOUTPUT.
- Add Parameters if you need to connect to a remote connection during definition
- If you are creating a Load Set adapter and your source data system has a fixed data structure, you may want to add Table definitions to the Adapter Area that match the source system's data structures and write code to use them to create target Table Descriptors at definition time.

Most shipped adapters use a Define-Time function.

The Define_Time_Function must have the following signature:

```
FUNCTION CreateLogFilePlannedOutput (pi_nCompanyID  IN NUMBER
  ,pi_nLSID       IN NUMBER
  ,pi_nLSVer      IN NUMBER) return BOOLEAN IS
```

The function's parameters take the following values:

- **pi_nCompanyID** see Getting Your Company ID
- **pi_nLSID** takes the Object ID of the Load Set definition
- **pi_nLSVer** takes the object version of the Load Set definition

Status_Recalc_Function

The system calls this function when a user modifies a Program, Business Area, or Data Mart. Although it is called the Status Recalculation function, the system automatically recalculates the installable status each time an object is modified, so you do not need to write a function for that purpose.

However, you can use this function to do anything your adapter requires when objects of your adapter type are modified.

Load Sets do not use the Status_Recalc_Function. The signature is different for Programs and Business Areas than it is for Data Marts. For Programs and Business

Areas, the system invokes the function when the user checks the object in or out. For Data Marts the system invokes the function when it computes the status—when the user checks in the Data Mart.

Shipped Data Mart and Program adapters use a Status_Recalc_Function, including SAS Export, Oracle Export, and Text Export Data Mart adapters and Informatica and BIP Program adapters.

See the following topics for signature details:

- Programs and Business Areas
- Data Mart Adapters

Programs and Business Areas

The Status_Recalc_Function for Program and Business Area adapters must have the following signature:

```
PROCEDURE RecalcInfaPgmStatus ( pi_sourceCdrNaming IN cdr_naming_version_obj_type,
  pi_vEvent IN VARCHAR2,
  pv_bothRefAndDef IN  cdr_df_naming_v.checked_out_flag_rc%type );
```

The function's parameters take the following values:

- **pi_sourceCdrNaming** is a parameter of table type CDR_NAMING_VERSION_OBJ_TYPE that contains object attributes. See the *Oracle Life Sciences Data Hub Application Programming Interface Guide* Reference Information section for details about the required attributes.
- **pi_vEvent** takes either **CHECKIN** or **CHECKOUT**, depending on the UI trigger event
- **pv_bothRefAndDef**: When the event is **CHECKOUT**, this parameter value is passed as **$YESNO$YES** if both the definition and instance are being checked out and **$YESNO$NO** if only the definition is being checked out.
  When event is **CHECKIN** this value is passed as **NA**.

Data Mart Adapters

The Status_Recalc_Function for Data Mart adapters must have the following signature:

```
PROCEDURE synchronizeDatamart ( pi_nDMCompanyId cdr_df_naming_v.company_id%TYPE,
  pi_nDMObjId     cdr_df_naming_v.obj_id%TYPE,
  pi_nDMObjVer    cdr_df_naming_v.obj_ver%TYPE,
  pi_vChildType   cdr_df_naming_v.object_type_rc%type);
```

The function's parameters take the following values:

- **pi_nCompanyID** see Getting Your Company ID
- **pi_nDMObjId** takes the Object ID of the Data Mart definition
- **pi_nDMObjVer** takes the object version of the Data Mart definition
- **pi_vChildType** takes the type of object that was modified, triggering the status recalc function. For example, if a user modifies a Table Descriptor in the Data Mart, then the child type is $OBJTYPES$TABLEDESCRIPTOR. See the Reference Information section in the *Oracle Life Sciences Data Hub Application*

Programming Interface Guide for information on how to look up the correct string for the object type from the lookup.

Object Installation Functions

Depending on the external system, you may need to write functions for use when a user installs a Load Set, Data Mart, Program, or Business Area. All object installation function names are stored in the tech_types table.

Each function should return `TRUE` in case of success and `FALSE` in case of a failure. All three of these functions must have the following signature:

```
function name_of_function (company_id in cdr_program_refs.company_id%TYPE,
  prref_id in cdr_program_refs.prref_id%TYPE,
  prref_ver in cdr_program_refs.prref_ver%TYPE, )
return boolean
```

The functions' parameters take values as follows:

- **pi_nCompanyID** see "Getting Your Company ID"
- **prref_id** see "Getting an Object's Prref_Id and Prref_Ver"
- **prref_ver** see "Getting an Object's Prref_Id and Prref_Ver"

See details for the following functions:

- Pre_Install_Function
- Install_Function
- Post_Install_Function

Pre_Install_Function

Your external system may require a function to be executed before a user installs a Load Set, Data Mart, Program, or Business Area.

Install_Function

Your external system may require a function to be executed when a user installs a Load Set, Data Mart, Program, or Business Area. The system calls this code immediately after the Pre-Install Function. There is no need to have both.

For example, the Informatica adapter uses the install function to create a folder in Informatica for every Informatica program being installed.

Post_Install_Function

Your external system may require a function to be executed immediately after a user installs a Load Set, Data Mart, Program, or Business Area.

Object Execution Functions and Procedures

Execution-related functions are stored in either the tech_types table or the adapter_areas table.

The following execution-related function names are stored in the tech_types table: Build_IDE_Cfg_Function, Build_Exe_Cfg_Function, Pre_Execution_Function, Execution_Function, and Post_Execution_Function.

The following execution-related function names are stored in the adapter_areas table: Currency_Function and Security_Recalc_Function.

Oracle LSH uses its Distributed Processing (DP) Server for executing jobs outside the Oracle LSH database server. For more information, see "Planning for Object Execution", "Planning Services" and the chapter on services in the *Oracle LSH System Administrator's Guide*.

For more information, see the following:

- Build_IDE_Cfg_Function
- Build_Exe_Cfg_Function
- Pre_Execution_Function
- Execution_Function
- Post_Execution_Function
- Currency_Function
- Security_Recalc_Function

Build_IDE_Cfg_Function

(For Visualization and Program adapters only.) This function is called when a user launches an integrated development environment (IDE) either from inside Oracle LSH or directly through the URL. Use it to do whatever is required for your system at that point; for example, create the URL for the IDE to be launched, download a file, or give the user the security access he or she needs in the external system.

You can use the input parameter pi_launchPref of the Build_IDE_Cfg_Function to launch one tool or mode of an IDE from the Oracle LSH Applications UI tab (the Business Area or Program properties page) and a different one from the Reports tab (for visualizations).

See "Planning Integrated Development Environment Adapters" for related information.

The build_ide_cfg_function must have the following signature:

```
Procedure proc_name(pi_nCompanyId   IN NUMBER,
  pi_nObjectId    IN NUMBER,
  pi_nObjectVer IN NUMBER,
      pi_nWorkAreaId    IN NUMBER,
  pi_nConfigId     IN NUMBER,
  pi_nConfigVer     IN NUMBER,
  pi_nPrrefId     IN NUMBER,
  pi_nPrrefVer     IN NUMBER,
  pi_launchPref   IN varchar2,
  po_allocateServiceInstance OUT VARCHAR2,
  po_launchData   OUT CDR_IDE_LAUNCH_DATA,
  po_isDBAccountAvailable OUT VARCHAR2);
  pi_vCdrUser     IN VARCHAR2,
  pi_vScemaName     IN VARCHAR2,
  pi_vPath     IN VARCHAR2,
```

The function's parameters take the following values:

- **pi_nCompanyID** see "Getting Your Company ID"
- **pi_nObjectId** takes the Object ID of the Business Area or Program instance.
- **pi_nObjectVer** takes the object version of the Business Area or Program instance.
- **pi_nWorkAreaId** takes the Object ID of the Work Area that contains the Business Area or Program instance.
- **pi_nConfigId** takes the Object ID of the Business Area or Program definition.
- **pi_nConfigVer** takes the object version of the Business Area or Program definition.
- **pi_nPrrefId** takes the PrrefID of the Business Area or Program instance; see "Getting an Object's Prref_Id and Pref_Ver". This parameter is not required but you can use the value to gain access to the Table Descriptors more quickly and improve performance.
- **pi_nPrrefVer** takes the PrrefID of the Business Area or Program instance; see "Getting an Object's Prref_Id and Pref_Ver". This parameter is not required but you can use the value to gain access to the Table Descriptors more quickly and improve performance.
- **pi_launchPref** Use this parameter if you need to open two different applications, each for a different mode—for example, if you need to launch a development environment and a visualization environment for the same adapter. For example, the OBIEE adapter launches the OBIEE Administrator's tool, which is on the Definer's PC, from the Business Area UI and OBIEE Presentation Services, which is on the OBIEE Server, from the Visualizations subtab of the Reports tab.

  The system passes one of the following values:

  – **LAUNCHFILE** The system passes this value (in uppercase) to the function if the user launches the IDE from the Applications tab.

  – **LAUNCHURL** The system passes this value (in uppercase) to the function if the user launches the IDE from the Reports tab.

  Your code can either handle these values differently or not.
- **po_allocateServiceInstance** must output `T` if a service instance is required for the adapter or `F` if it is not; see "Planning Services".
- **po_launchData** is a collection that outputs attribute values as follows:

  – **launch_mode** can have a value of either `URL` or `FILE`, depending on whether the adapter needs to launch a URL or open a file. The value of this attribute determines which of the other attributes are required. You must write the code to make the function return the value that is appropriate for your adapter.

  If launch_mode is set to FILE, the system launches the IDE on the Definer's PC by pushing a file with extension .cdz.

  > **Note:**
  >
  > Oracle LSH supports only IDEs that are installed on the Definer's PC.

If launch_mode is set to URL, the system forwards the IDE launch request to the URL, which is another attribute of the CDR_IDE_LAUNCH_DATA collection.

- **url** is required if the launch_mode value is `URL`. It must provide the actual URL the adapter is to launch.

> **Note:**
>
> If you are creating this adapter for use in other companies or locations, you can use the Execution Command field of the service type definition for the purpose of collecting the URL. The field is available because IDE technology types do not require an execution command. This value is stored in the EXECUTION_COMMAND column of the CDR_SERVICES table (varchar2(200) BYTE).

- **blob_elements** is a collection of BLOBs. It is required if the launch_mode value is `FILE` and the file is binary. It outputs the actual file(s) that must be downloaded to the client.
- **blob_filenames** is a collection that is required if the launch_mode value is `FILE` and the file is binary. It outputs the name of the file(s) that must be downloaded to the client. There must be a blob_filename value for each blob_element.
- **clob_elements** is a collection of CLOBs. It is required if the launch_mode value is `FILE` and the file is character-based. It outputs the actual file(s) that must be downloaded to the client.
- **clob_filenames** is a collection that is required if the launch_mode value is `FILE` and the file is character-based. It outputs the name of the file(s) that must be downloaded to the client. There must be a clob_filename value for each clob_element.
- **col_comp_list** is a collection that is required if you need to download a file to the IDE or pass any other values to the IDE when a user launches the IDE. Specify one or more argument/value pairs; for example, `"_comp002_"`, `"<filename>"` where the filename is the file to be downloaded. The argument name and value must match the those in the cdrconfig.xml file; see "Planning Integrated Development Environment Adapters".
- **col_sub_dirs** is a collection that is required for users to view outputs generated by the Program they are defining in the IDE. This attribute must name the directories into which the system should put these outputs; see "Creating Subdirectories on IDE Computers".

- **po_isDBAccountAvailable** outputs `T` if a database account is required by the adapter or `F` if it is not. For example, the SAS adapter requires a database account to reconnect to the Oracle LSH database during IDE launch.
- **pi_vCdrUser** takes the Oracle LSH user ID of the person launching the visualization tool.
- **pi_vSchemaName** takes the Oracle LSH database account user ID of the person launching the visualization tool.
- **pi_vPath** takes the full path of the object:

```
user_db_account/domain_name/app_areaname/wa_name/object_name/vversion_number
```

for example:

```
John_Smith_DB/SmokeTestDomain/SmokeTestAA/ SmokeTestWA/Study55698_BA/v3
```

Build_Exe_Cfg_Function

This function is not called by the system. Use it only if you need an additional function that another function can call.

For example, in the SAS adapter, the pre-execution function calls a procedure buildSasExeContent to generate dynamic components of a SAS configuration for execution. The binary components are returned in a list of BLOBs and the ASCII components are returned in a list of CLOBs.

Pre_Execution_Function

Your adapter may require code to be run immediately before executing a Load Set, Data Mart, or Program. For example, a pre-execution procedure might check connectivity with the external system, prepare scripts for downloading to the Distributed Processing Server, and, for Load Sets, verify that source data structures and Oracle LSH target Table instances are compatible. The Pre_Execution_Function requires the following signature:

```
procedure <procname>(pi_nJobId in cdr_jobs.job_id%type)
```

pi_nJobId takes the jobID of the current object execution.

Some shipped adapters use a Pre_Execution_Function, including: SAS, Text, and Oracle Clinical Data Extract SAS View Load Sets; Oracle and SAS Data Marts.

Execution_Function

This function is responsible for coordinating the execution of a Load Set, Data Mart, or Program. It must call several public APIs. These are documented in the *Oracle Life Sciences Data Hub Application Programming Interface Guide*. See also "Planning for Object Execution".

The execution function must do the following:

1. Fetch the current job ID. Use the public API CDR_PUB_EXE_RUNTIME.GETCURRENTLYEXECUTINGJOBID.
2. If required, build a collection of BLOBs or CLOBs and the corresponding BLOB or CLOB file names for the files that the Oracle LSH Distributed Processing (DP) Server must download when it runs the job, and call public API CDR_PUB_EXE_EXTERNAL.CREATETEMPLOBS to upload the BLOBs or CLOBs to a temporary table. This constitutes the job payload.
3. Build an XML file to send to the DP Server so that the DP Server can start executing the job. To create the XML file, call public API CDR_PUB_EXE_EXTERNAL.GENERATEXMLPAYLOAD. This constitutes the XML payload; see "Example 2-2".
4. Start the job execution by calling public API CDR_PUB_EXE_EXTERNAL.SENDJOB.

The DP Server then receives the XML payload, downloads the job payload, and starts the job.

5. The function must wait for the job to complete. Call public API CDR_PUB_EXE_EXTERNAL.WAITFORFINALSTATUS.

   The Wait For Final Status API returns 1 if the job completes without warnings, 2 if the job completes with warnings, and 3 if the job fails.

6. If the job succeeds, return 0; else return 1.

   If the job produces one or more outputs, the DP Server loads the BLOB or CLOB file(s) into a temporary table. Your postexecution function should retrieve these files; see "Post_Execution_Function".

Example 2-1 XML Payload Required File Structure

```
<?xml version="1.0" ?>
<EXEJOB EXEJOB_VERSION="1.0">
<JOB ID="797120006" TYPE="exe">
<SURROGATEJOBID>797120006</SURROGATEJOBID>
<PRREFID>42580001</PRREFID>
<CONFIGID>0</CONFIGID>
<WORKDIR>/user/oracle/sas92/SasWork/ip1dv102</WORKDIR>
<PROGRAM>/user/oracle/sas92/sasNormal</PROGRAM>
<RUNSCRIPT>runSasJob</RUNSCRIPT>
<OUTPUTPATH>Output</OUTPUTPATH>
<PRIORITY>$JOBPRIORITIES$NORMAL</PRIORITY>
<SCHEMA>ZZ_CDR_SI_610001</SCHEMA>
<USERID>797120006</USERID>
<SUBDIRS><DIR NAME="SOURCE" /><DIR NAME="Output" /><DIR NAME="LSH_RS" />
</SUBDIRS>
</JOB>
</EXEJOB>
```

The Execution_Function requires the following signature:

```
function <funcname>return number;
```

Example 2-2 XML Payload

For technologies that run outside the database and therefore use the OWB operator CdrService_1, you can call API CDR_PUB_EXE_EXTERNAL.GENERATEXMLPAYLOAD to produce a default XML payload file. If your technology requires additional functionality—for example, passing additional values to the job—your execution function can produce an XML file with the information your adapter requires. However, your execution function must use the same structure.

The required XML payload file structure is:

```
<?xml version="1.0" ?>
<EXEJOB EXEJOB_VERSION="1.0">
<JOB ID="id_of_currently_executed_job" TYPE="exe">
<SURROGATEJOBID>id_of_master_job_or_same</SURROGATEJOBID>
<PRREFID>prref_id_of_the_executing_program</PRREFID>
<CONFIGID>0</CONFIGID>
<WORKDIR>/user/oracle/work_directory</WORKDIR>
<PROGRAM>/user/oracle/execution_command_location/execution_command_script</PROGRAM>
<RUNSCRIPT>entry_point_run_script</RUNSCRIPT>
<OUTPUTPATH>Output_path</OUTPUTPATH>
```

```
<PRIORITY>$JOBPRIORITIES$NORMAL</PRIORITY>
<SCHEMA>ZZ_account_from_service_instance</SCHEMA>
<USERID>(optional)</USERID>
<SUBDIRS><DIR NAME="first_subdirectory" /><DIR NAME="another_subdirectory" /><DIR
NAME="yet_another_subdirectory" />
</SUBDIRS>
</JOB>
</EXEJOB>
```

Additional information:

- **Job ID** and **Surrogate Job ID**. If the job is has only a single process, these two IDs are the same. If there is a master job and a subjob, the Job ID is for the master job's ID and the Surrogate Job ID is for the current subjob.
- **Prrefid**. The Prrefid of the object being executed. You can use API CDR_PUB_EXE_RUNTIME.GETJOBINFO to get this ID from the CDR_SUBMISSIONS table.
- **Configid**. The object ID of the object being executed.
- **Workdir**. This is the location set up under the DP Server directory for jobs of this type and named as the Root Directory in the service definition. It is stored in the ROOT_DIRECTORY column of the CDR_SERVICES table. The DP Server creates a subdirectory here for each job with the job ID as a name, containing the files required for the job.
- **Program**. (Optional) This value is stored in the PROGRAM column of the CDR_SERVICES table.
- **Run Script**. Your execution function must generate a script for each job that includes the service instance assigned to the job and provides the actual starting point for the job execution.
- **Output Path**. Enter the path for the DP Server Home/log directory.
- **Priority**. The priority requested by the user for the service instance. Possible values are: $JOBPRIORITIES$NORMAL, $JOBPRIORITIES$HIGH, or $JOBPRIORITIES$LOW.
- **Schema**. The ZZ% account of the service instance. You can use the SERVICE_INSTANCE_ID to get the ZZ account from the table CDR_SERVICE_INSTANCES.
- **User ID**. This value is not required. It is used only by one shipped adapter.
- **Subdirectories**.

See the *Oracle Life Sciences Data Hub Application Programming Interface Guide* for information on execution-related APIs.

Post_Execution_Function

Your adapter may require a procedure to run after the execution of a Load Set, Data Mart, or Program. For example:

- If the execution produces outputs, the DP Server puts these files in a temporary table. You can use the view CDR_TEMP_BLOBS_V to get a list of the outputs uploaded by the DP Server and call the public API CDR_PUB_EXE_EXTERNAL.UPLOADBLOBOUTPUT to upload the outputs.

- You can create a procedure to search the log file for specific information after execution.

The Post_Execution_ Function requires the following signature:

```
procedure <procname>(pi_nJobId in cdr_jobs.job_id%type)
```

pi_nJobId takes the jobID of the current object execution.

> **Note:**
>
> If any of these programs returns an error the job returns an error. If any of these programs returns a warning then the job returns a warning unless another part of the job generated an error.

Currency_Function

(Optional; applies to Load Sets only. Oracle LSH does this automatically for Programs and Data Marts.) Use this function to determine the currency of the data in the external system. If you define a currency funtion, the system automatically invokes it when the job is executed and uses it to determine whether to run the job.

The currency function must return the source data currency. Oracle LSH job processing logic then determines whether the currency of the previous successful job was the same as the current job. If so, it marks the current job as duplicate and stops the job (unless the Force Execution flag is set to **Yes** by the Force Execution system Parameter). If the currency values are different, then the system proceeds to execute the job; see "Planning for Object Execution".

One of the system Parameters available for submitting a job is Currency_Type, whose value indicates whether the job will take the most current data or use a data snapshot. The default is to use the most current data. If there is no currency function available and the parameter value is for current data, the system processes the most current data.

The following shipped Oracle Clinical Load Set adapters use a currency function: Data Extract Oracle Views, Data Extract SAS Views, Labs, and Global Metadata.

This procedure has the following required signature:

```
PROCEDURE getCurrency(pi_nCompanyID        IN NUMBER
  ,pi_nPrrefId          IN NUMBER
  ,pi_nPrrefVer         IN NUMBER
  ,pi_cRunParamNVPair   IN CDR_NAME_VALUE_PAIR_COLL
  ,pi_cSysParamNVPair   IN CDR_NAME_VALUE_PAIR_COLL
  ,po_cCurrencyListColl OUT NOCOPY CDR_CURRENCY_LIST_COLL);
```

The procedure's parameters take values as follows:

- **company_id** see "Getting Your Company ID"
- **prref_id** see "Getting an Object's Prref_Id and Prref_Ver"
- **prref_ver** see "Getting an Object's Prref_Id and Prref_Ver"
- **pi_cRunParamNVPair** is a collection with a name,value pair for each runtime Parameter for the job that is passed to the currency function. For example, Oracle

Clinical Labs Load Sets pass the remote location and lab name to the adapter's currency function.

- **pi_cSysParamNVPair** is a collection with a name,value pair for each predefined system Parameter for the job that is passed to the currency function. System Parameters include currency type, job priority, force execution, and more. For a complete list and descriptions, see the *Oracle Life Sciences Data Hub User's Guide*.
- **po_cCurrencyListColl** this output parameter is a collection of type CDR_CURRENCY_LIST_COLL, which is a table of CDR_CURRENCY_OBJ_TYPE that has the following elements:
  - VCCURRVALUE VARCHAR2(4000)
  - DTCURRVALUE DATE
  - NUMCURRVALUE NUMBER

  The adapter can return how current the data in the remote system is by either returning a character, date or number currency or any combination of the three. For example, the Oracle Clinical Labs currency function fetches the max date value from the tables labs, lab_range_subsets and ranges.

Security_Recalc_Function

You may need to write a procedure to synchronize security with the external system; see "Synchronizing Security with Integrated Environments". If so, set the Security_Recalc_Flag_RC for the Adapter Area to `$YESNO$YES` and write a procedure with the following signature:

```
Procedure RecalcDiscovererSecurity (pi_nCompanyId     IN NUMBER,
  pi_nObjId       IN NUMBER,
  pi_nObjVer      IN NUMBER);
```

The function's parameters take the following values:

- **pi_nCompanyId** see "Getting Your Company ID"
- **pi_nObjId** takes the Object ID of the object instance the user is trying to use.
- **i_nObjVer** takes the object version number of the object instance the user is trying to use.

Planning Parameters and Parameter Sets

You must plan parameters and parameter sets to get information from the user.

If you need information from the user during Load Set, Data Mart, Program, or Business Area definition, installation, or execution, you must do the following:

- Define a Parameter Set in the Adapter Area with one of three required names and usage settings, depending on whether the Parameter value is required in order to define or install the object in general, to define Table Descriptors for the object, or to collect information at runtime.
- When you run the Create Adapter Area API, set a flag to indicate that this type of Parameter Set exists (although you must create the Adapter Area before you can create the Parameter Set).

- Define a Parameter for each piece of information you need to collect, and create an instance of it in the appropriate Parameter Set.

The system then displays the Parameters you define in the proper place in the user interface and uses their values at the correct time.

> **Note:**
>
> If you are developing an adapter for use in other locations and need to collect information about the Oracle Life Sciences Data Hub installation or the external system installation, you can use the Details field of the service defintion; see Planning Services.

For more information, see the following:

- General Define-Time Parameters
  If you need information from the user during Load Set, Data Mart, Program, or Business Area definition, create a Parameter Set named `PARAMETERSET_LOADSETLEVEL_DEF` and set its Usage attribute to `DEFINITION`.
- Table Descriptor Define-Time Parameters
  (Load Set adapters only) Load Sets' Table Descriptors are based on data structures in the external system.
- Runtime Parameters
  If you need information from the user when he or she runs the Load Set, Data Mart, or Program, or launches a visualization tool, create a Parameter Set named `PARAMETERSET_LOADSETLEVEL_RUN` and set its Usage attribute to `EXECUTION`.

General Define-Time Parameters

If you need information from the user during Load Set, Data Mart, Program, or Business Area definition, create a Parameter Set named `PARAMETERSET_LOADSETLEVEL_DEF` and set its Usage attribute to `DEFINITION`.

Oracle Life Sciences Data Hub displays Parameters in this Parameter Set as attributes in the Load Set, Data Mart, Program, or Business Area Properties page in the user interface.

For example, the following shipped adapters have these attributes:

- **SAS and Text Load Sets**: Save Input File (Yes or No)
- **All Data Marts**: File Name (for the Data Mart's output file)
- **Text Data Marts**: Mode (Delimited or Fixed) and Filename Extension (.csv or .txt)

If you need Parameters whose values are used during Load Set, Data Mart, Program, or Business Area installation, you can define them in the `parameterset_loadsetlevel_def` Parameter Set.

Table Descriptor Define-Time Parameters

(Load Set adapters only) Load Sets' Table Descriptors are based on data structures in the external system.

If user input is required to define these Table Descriptors, create Parameters to collect this information and create an instance of each Parameter in a Parameter Set with the required name: `parameterset_operatorlevel` and set its Usage attribute to `OPERATOR`.

Oracle Life Sciences Data Hub creates attributes in the object definition user interface that correspond to the Parameters you define in the `parameterset_operatorlevel` Parameter Set. In the user interface, these attributes appear in the same location as the `parameterset_loadsetlevel_def` attributes, but the system uses their values at different times.

For example, the following shipped adapters have the following Table Descriptor attributes:

- **Oracle Tables and Views Load Sets**: Remote Location and Database Schema
- Some **Oracle Clinical Load Sets**: Remote Location and Study Name

Note:

If your adapter requires a Remote Location attribute, after you have defined the adapter you must also define at least one remote location in the Oracle Life Sciences Hub user interface; see*Oracle LSH System Administrator's Guide* for further information.

Runtime Parameters

If you need information from the user when he or she runs the Load Set, Data Mart, or Program, or launches a visualization tool, create a Parameter Set named `PARAMETERSET_LOADSETLEVEL_RUN` and set its Usage attribute to `EXECUTION`.

Oracle Life Sciences Data Hub displays Parameters in this Parameter Set as Parameters in the Parameters tab in the Load Set, Data Mart, Program, or Business Area Properties page and in the Load Set, Data Mart, or Program's Execution Setup.

For example, the following shipped adapters have the following runtime attributes:

- **SAS Load Sets**: Dataset Filename and BLOB ID (Temporary)
- **Text Load Sets**: Data File Name, Data Format, Delimiter Character, Enclosing Character, Initial Records to Skip, Maximum Allowed Errors, Temp LOB ID, Date Format
- **Oracle Tables and Views Load Sets**: Remote Location
- **SAS Data Marts**: Mode and Zip Result
- **Oracle Export Data Marts**: Compress and Statistics
- **Text Data Marts**: Zip Results, FirstRow Desc, Operating System, Separating Character, Use Enclosing Character, Enclosing Character

Note:

If you are creating an adapter that must upload files, define a Parameter with its Parameter Type set to either BINARY_FILE or TEXT_FILE, as appropriate, and create an instance of it in the runtime Parameter Set. Create a second Parameter called TMP_BLOB_ID, for example, and create an instance of it in the same Parameter Set. Write code to upload the file to a temporary location and store the ID for the file as the value of TMP_BLOB_ID. Use this value in your pre-execution or execution function.

Adding Lookup Values

You must extend certain lookups. You can also enable your customers to add lookup values in their own environment.

Extend the following lookups:

- **CDR_TECH_TYPES**. At least one new technology type is required for your adapter. Enter the same value that you enter in the pio_techTypeRow.TECH_NAME_RC parameter of the Create Technology Type API. This value is used internally.
- **CDR_SERVICE_TYPES**. One service type per technology type is required. This value appears in the user interface for defining services.
- **CDR_FILE_TYPES**. Your adapter may need need a new file type; for example, for source code in Program or Business Area adapters. Add the file type to this lookup.

For instructions on extending lookups in your environment, see the chapter on lookups in the *Oracle Life Sciences Data Hub System Administrator's Guide*.

Provide instructions to each company using the adapter for adding the lookup values required for your adapter in their own environment.

Planning Planned Outputs

Data Mart adapters may require predefined Planned Outputs to support the actual data file produced by running the Data Mart.

Program adapters may required predefined Planned Outputs to support report outputs.

Data Mart, Program, and Load Set adapters may all need predefined Planned Outputs to serve as placeholders for other outputs created when they are executed; for example, log and error files.

To automatically create a fixed Planned Output definition for every object of your adapter's object type (Data Mart, Program, or Load Set), call public API CDR_PUB_DF_PLANNED_OUTPUT.CREATEPLANNEDOUTPUT in your define_time_function.

Planning Data Structures

If you are creating a Load Set adapter and your source data system has fixed data structures, you can define Tables in your Adapter Area with the same structure.

See Call the Create Table API. You then write your adapter code to create Load Set Table Descriptors based on those Tables rather than requiring connection to the remote database or requiring manual definition by users.

The Oracle Clinical Load Set adapters that import data from fixed Oracle Clinical tables do this, including Design and Definition, Labs, Randomization, and Study Data.

Planning for Object Execution

Programs, Load Sets, and Data Marts are all executable objects and an adapter of these types must handle the object execution. Business Area and Program IDE adapters may also require execution functionality.

Some execution functionality is built in to Oracle Life Sciences Data Hub, but each adapter requiring execution must have a custom execution function, a custom service type, and a custom execution command.

> **Note:**
>
> Processing engines that run on the database do not use the DP Server.

For more information, see the following:

- Execution Process
- Execution Command

Execution Process

The execution process includes:

1. The user submits a Load Set, Program, or Data Mart for execution.
   - The system calls the OWB operator specified for the technology type and creates an OWB audit task, which OWB tracks in order to report the job's status. OWB returns control to Oracle LSH Runtime (called "the system" here).
   - The system generates a job ID and assigns a service instance to the job.
2. The system calls the pre-execution function specified in the technology type, if any, and then calls the execution function.
3. You must write an execution function to do the following; see "Execution_Function" for details.
   a. Build a collection of files, if required
   b. Output an XML message with information about the job (the XML payload)
   c. Call the Send Job public API

4. The Oracle LSH listener detects the XML message produced by the Execution_Function, dequeues the message, and sends the XML message to the DP Server to run the job in the external processing engine using the assigned service instance.

   The service instance is the account that the job uses to connect back to the Oracle LSH database to read from sources and write to targets, if any.

   The DP Server creates a subdirectory with he job ID as a name in the work directory defined under the DP Server directory for jobs of this type and puts all the files required for the job in this subdirectory.

   The DP Server makes two UNIX shell variables available to the processing engine:

   - $CDRJOBSCHEMA holds the value of the service instance account, which is extracted from the XML message; for example, <SCHEMA>ZZ_CDR_SI_250001</SCHEMA>
   - $CDREXEPASS holds a random password assigned to the account for that job. The password can never be reused.

5. The service instance runs the execution command. There is a single execution command, which you must write, for all jobs of this type; see "Execution Command".
6. The DP Server job processor uses the information in the XML message to connect back to the Oracle LSH database. Once connected, it downloads all the necessary data, including the execution function(s), into a target directory identified in the XML message.
7. The job processor then executes the job-specific script by spawning a new operating system-level process.
8. This spawned process runs the processing engine, which in turn carries out the actual job execution using the downloaded job input data.
9. After the spawned external process completes, the job processor connects back to Oracle LSH and uploads any output results into Oracle LSH. The service instance is released.
10. Meanwhile, the execution function waits for the job to complete, using the Wait for Final Status public API as soon as it calls the Send Job API.

    If the job produces one or more outputs, the DP Server loads the BLOB or CLOB file(s) into a temporary table. Your postexecution function should retrieve these files; see "Post_Execution_Function".
11. The system calls the postexecution function.
12. OWB reports the final job status. See the *Oracle Life Sciences Data Hub User's Guide* for a description of job and execution statuses.

Execution Command

For processing engine technology types, you must write an execution command file.

The execution command invokes the external processing system. It must be contained in a file of a type appropriate for the operating system; a shell script for UNIX or a command file for Windows.

In most cases, the execution command must provide a mechanism to pass the service instance to the external processing engine which can then use the service instance's connection credentials to connect back to the Oracle Life Sciences Data Hub database.

To help ensure the proper storage and display of non-English character data in Oracle Life Sciences Hub, the processing engine should use UTF8 character encoding. If you can enforce this in the execution command, do so.

As with the shipped adapters, at each Oracle Life Sciences Data Hub installation the user must move the execution command file into the DP Server directory created by the user, or a subdirectory of it, and enter the actual path of the file in the service location definition in the Oracle Life Sciences Hub user interface (see the *Oracle Life Sciences Data Hub System Administrator's Guide*). The system picks up the location from the service location definition when a user submits a job.

If you are creating this adapter for use in other companies or locations, you must include instructions for system administrators similar to those in the *Oracle Life Sciences Data Hub Installation Guide* to move the file and edit it to add whatever information your adapter may require about the location; for example:

- the Oracle SID
- the location of the technology server
- the location of Oracle setup script `coraenv`
- a variable holding the path to any external command invoked inside the execution command; for example, if the execution command invokes a Java executable, the path to the Java executable must be set correctly in the path variable.

Planning Security

The security required for your adapter depends on the interaction with the external system that it requires.

Note the following:

- Data Mart adapters and processing engine adapters for Programs require no security coordination with the external system. The user must have proper Oracle LSH security privileges in order to define and run the Data Mart or run the Program. The DP Server automatically provides an account and unique password for each job through the service instance to allow the processing engine to connect to the Oracle LSH database to process data; see Planning for Object Execution.
- Load Set adapters may require security coordination with the external system; for example, it may be necessary for users to have a user account with the same name in both systems. File-based Load Sets require access to the files to be loaded. Database-based Load Sets may require additional security.
- Program and Business Area IDE adapters generally require the most complex security coordination between the two systems; see Synchronizing Security with Integrated Environments.

Planning Integrated Development Environment Adapters

There are a number of things your Definers may need to be enabled to do in an integrated development environment (IDE). You must consider these needs when planning IDE adapters.

Business Area and Program adapters may require that Oracle Life Sciences Data Hub Definers be able to do the following in a development environment that is integrated with Oracle Life Sciences Hub:

- Launch the integrated development environment (IDE) from within Oracle Life Sciences Hub either from the Program or Business Area Properties page (for development) or from the Reports tab, Visualizations subtab (for viewing data visualizations), and then see appropriate Oracle Life Sciences Hub data in the IDE.
- Download and upload files between Oracle Life Sciences Hub and the IDE—or the adapter may need to do this automatically.
- Log in directly to the external system and gain access to Oracle Life Sciences Hub objects and data.Oracle Life Sciences Hub allows this access only for adapters that view but do not modify Oracle Life Sciences Hub data. In this case, you can use the generic visualization adapter; see Using the Generic Visualization Adapter.

For more information, see the following:

- IDE Launch Process from Within Oracle Life Sciences Data Hub
- Synchronizing Security with Integrated Environments
- Adding Source Code Types
- Planning Navigation in the External System to Business Areas
- Editing cdrconfig.xml
- Creating Subdirectories on IDE Computers

IDE Launch Process from Within Oracle Life Sciences Data Hub

When a user launches an IDE from within Oracle Life Sciences Hub, the system:

- Calls the Build_IDE_Cfg_Function to launch the IDE, download files, if necessary, and whatever else you have coded it to do.
  Oracle Life Sciences Hub makes a distinction—by passing a different value to the Build_IDE_Cfg_Function—between IDEs launched from the Reports tab, through which Consumers view data visualizations, and IDEs launched from Program and Business Area pages, through which Definers develop Programs and Business Areas. Consumers can see blinded data in an IDE if they have the required privileges in Oracle Life Sciences Hub, but Definers can never see blinded data when they launch the IDE from a Program or Business Area.
- Calls the Security_Recalc_Function and passes values to it for the company ID, object ID, and object version of the object—Program or Business Area—for which the IDE is being launched. You can use these to get the prrefid of the Program or Business Area to set the context for data security; see Getting an Object's Prref_Id and Prref_Ver and Establishing Context .

- Assigns a service instance to the IDE session. The service instance is the account that the job uses to connect back to the Oracle LSH database. It remains allocated to the Program or Business Area until it is explicitly released by a Checkin or Undo Checkout user action.
  The service instance assigned to the Program or Business Area during IDE launch is associated with the user's database account, so if the user logs in using their database account, the system automatically sets up the sys context so that they can read the data in Table instances mapped to the Program or Business Area's Table Descriptors.

Synchronizing Security with Integrated Environments

Your adapter must synchronize security with the external system:

- **Object Security**: Users should be able to access only the appropriate Oracle Life Sciences Data Hub objects (Program or Business Area) from the external system and have only the appropriate privileges on those objects.
- **Data Security**: Users should be able to see data only in Table instances that are mapped to the object's Table Descriptors, and should be able to view only data in appropriate currency and blinding states.

Both types of security must be enforced both when the user launches an IDE from within Oracle Life Sciences Hub and when the user logs in directly to the external system.

The Oracle Life Sciences Hub security system enforces object security within Oracle LSH through user accounts. Viewing data requires an Oracle Life Sciences Hub database account mapped to a user account. Your adapter can use one or more Oracle Life Sciences Hub user accounts to take advantage of this functionality.

Within Oracle Life Sciences Hub, the system enforces that users can see data only in Table instances mapped to the current object. You can use the Tracking API and Tracking Table to enforce the same restriction in the external system.

See the *Oracle Life Sciences Data Hub Implementation Guide* and the *Oracle Life Sciences Data Hub System Administrator's Guide* for information on Oracle Life Sciences Hub security.

For details, see the following:

- Enforcing Security on Corresponding External Entities
  You may need to create entities in the external system that correspond to Oracle Life Sciences Data Hub objects.
- Establishing Context
  An **application context** is a set of name-value pairs that Oracle Database stores in memory.
- Tracking and Removing Object Access
  Program and Business Area adapters allow users to launch an integrated development environment (IDE) from Oracle Life Sciences Data Hub to modify an Oracle Life Sciences Hub Program or Business Area's source code.

Enforcing Security on Corresponding External Entities

You may need to create entities in the external system that correspond to Oracle Life Sciences Data Hub objects.

For example, the Informatica adapter creates a different Informatica Folder for each Oracle Life Sciences Hub Informatica Program, and the OBIEE adapter creates a different OBIEE Subject Area for each OBIEE Business Area. You can then use the external system's security to enforce access to the appropriate external system entities.

In the case of Informatica, the Build_IDE_Cfg_Function interacts with Informatica through the DP Server to assign privileges to the user for the Folder specific to the Program. Alternatively, you can use the Security_Recalc_Function to synchronize security in your adapter.

You can use the Tracking API and Tracking Table to maintain a record of the object the same user accessed during his or her previous session in the IDE and then use the security APIs of the external system to remove access to that object in the current session; see Tracking and Removing Object Access.

Establishing Context

An **application context** is a set of name-value pairs that Oracle Database stores in memory.

An **application context** is a set of name-value pairs that Oracle Database stores in memory. The application context has a label called a namespace. Inside the context are the name-value pairs (an associative array). An application can use the application context to access session information about a user, such as the user ID or other user-specific information, and then securely pass this data to the database. You can then use this information to either permit or prevent the user from accessing data through the application.

For Oracle Life Sciences Data Hub, you need a name-value pair for each of the following:

- User ID
- User password
- PrrefID of the Program or Business Area

You can set the context as follows:

- If no login is required when the user launches the external system, set the sys_context by writing a function.
- Establish the context with the database logon trigger. The adapter doesn't need to do anything specifically for allowing the user access to the correct data. The Oracle Life Sciences Hub logon trigger uses the user's database account to determine the user's privileges and enables the sources and targets so that the user can view LSH data from the IDE.

  Note:

  For information on application context functionality, logon triggers, and more, see the *Oracle® Database Security Guide* 11g Release 2 (11.2) at `https://docs.oracle.com/cd/E11882_01/server.112/e10575/toc.htm`.

- If you are using the Generic Visualization adapter, see Initializing the Business Area Instance.

Tracking and Removing Object Access

Program and Business Area adapters allow users to launch an integrated development environment (IDE) from Oracle Life Sciences Data Hub to modify an Oracle Life Sciences Hub Program or Business Area's source code.

You must ensure that Definers can modify only Programs and Business Areas on which they have Modify privileges in Oracle Life Sciences Hub by using the external system's APIs or another security mechanism.

One approach is to ensure that when users launch the IDE, which they can do only through a Program or Business Area, that they have access only to that particular Program or Business Area in that session, even if they have access to other Programs or Business Areas of the same IDE type and have worked on them in the past.

For example, if the external system has entities that correspond to Oracle Life Sciences Hub Programs or Business Areas (such as Informatica Folders or OBIEE Subject Areas) and public APIs that grant and revoke access to these entities, you can design your adapter so that when the user launches the IDE, the adapter calls the external system's API to:

- grant access to the entity corresponding to the Program or Business Area
- revoke access to previously granted entities, if any

In order to revoke access to previously granted entities, you must maintain a record of each user's IDE launches and the Programs, Business Areas, and the corresponding external entities to which each user was granted access. Oracle Life Sciences Hub has a public API, CDR_PUB_EXE_EXTERNAL.TRACKLAUNCHIDE, and a table, CDR_IDE_LAUNCH, for this purpose; see Tracking API and Tracking Table.

In your Build_IDE_Cfg_Function you can call CDR_PUB_EXE_EXTERNAL.TRACKLAUNCHIDE and the external system's APIs to grant and revoke access to the appropriate entities in the external system.

In addition, in the case of Programs, your function may need to determine whether the Program being launched uses source code shared from another Program and if so, grant read access to the corresponding external entity for that Program too. You then also need to be sure to revoke access from all entities included in the previous launch. They have the same IDE_LAUNCH_ID.

> **Note:**
>
> No Business Area types currently included with Oracle Life Sciences Hub use shared source code, but it is theoretically possible.

See the following reference information:

- Tracking API
- Tracking Table

Tracking API

Use the public API CDR_PUB_EXE_EXTERNAL.TRACKLAUNCHIDE to pass information to the tracking table about the current user and IDE launch.

You can set a parameter to indicate whether to delete the record of the user's previous IDE launch from the table or not.

The API's signature is:

```
(p_api_version IN NUMBER
,p_init_msg_list IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_commit IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_validation_level IN NUMBER default CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL
                    ,x_return_status OUT NOCOPY VARCHAR2
                    ,x_msg_count OUT NOCOPY NUMBER
                    ,x_msg_data OUT NOCOPY VARCHAR2
                    ,pi_IdeLaunchColl IN CDR_IDE_LAUNCH_OBJ_COLL
    ,        pi_DelPrevEntry IN VARCHAR2
);
```

The nonstandard parameters for this API are:

pi_IdeLaunchColl

cdr_ide_launch_obj_coll is a collection that is a table of cdr_ide_launch_obj_type, with the following attributes:

- COMPANY_ID. NUMBER(6)
- IDE_LAUNCH_ID. NUMBER(22)
- PRREF_ID. NUMBER(22)
- PRREF_VER. NUMBER(7)
- SHARED_FLAG_RC . VARCHAR2(30)
- EXT_SYS_ENTITY. VARCHAR2(4000)
- LAUNCH_USER_ID. NUMBER(15)

pi_DelPrevEntry

Set to **Y** to delete the record of the current user's previous IDE launch. Set to **N** to maintain the complete IDE launch history for the user.

Tracking Table

If your adapter uses the TRACKLAUNCHIDE API, the internal table CDR_IDE_LAUNCH contains a record for each Program or Business Area required for the current IDE launch.

If the Program being launched uses shared source code included in another Program, the API creates a record for both Programs and gives both records the same IDE Launch ID.

The table has the following columns:

COMPANY_ID

NUMBER (6). To get the company ID, use CDR_PUB_DEF_FACTORY_UTILS.GETCOMPANYID.

IDE_LAUNCH_ID

NUMBER (22). This value is generated from a sequence every time the user launches the IDE and the TRACKLAUNCHIDE API is called.

PRREF_ID

NUMBER (22). Prref ID of the Program or Business Area from which the IDE is being launched or the one containing shared source code. See Getting an Object's Prref_Id and Prref_Ver.

PRREF_VER

NUMBER (7). Prref version of the Program or Business Area from which the IDE is being launched or the one containing the shared source code. See Getting an Object's Prref_Id and Prref_Ver.

SHARED_FLAG_RC

VARCHAR2 (30). If **N**, the Program is the Program being launched. If **Y**, the Program contains shared source code required for the Program being launched.

Note:

Normally Business Area adapters do not use shared source code and always have this flag set to **N**.

EXT_SYS_ENTITY

VARCHAR2 (4000). The entity in the external system which maps to the Oracle LSH Program or Business Area. For example, each Oracle Life Sciences Data Hub Informatica Program corresponds to a Folder in Informatica. If this relationship is required, your adapter must track it.

LAUNCH_USER_ID

NUMBER (15). ID of the user who is launching the IDE.

Adding Source Code Types

If you are creating a Program or Business Area adapter, you may need to create one or more new Source Code types and a new file type for each new Source Code type.

Before you can create a source code type you must:

- Create a technology type by calling the Create Technology Type API; see Creating a Technology Type. Enter the Tech Type ID in the TECH_TYPE_ID column for the source code type.

- Add the file type you need for your adapter source code by extending the CDR_FILE_TYPES lookup; see Adding Lookup Values. Enter the file extension in the DEFAULT_EXTN column for the source code type. Do not include the dot (.) before the letters in the extension.

You then create a a new Source Code type to link your technology type with your file type. To create a new Source Code type, call the procedure cdr_srccode_types_m.insertRow:

```
PROCEDURE InsertRow( pRecord   in out nocopy CDR_SRCCODE_TYPES%rowtype );
```

The columns in the Source Code Types table include the following. See the section on Source Code in the "Defining Programs" chapter of the *Oracle Life Sciences Data Hub Application Developer's Guide* for information about these Source Code attributes.

- **Tech Type ID**. Enter the ID of your technology type. This is generated by Oracle LSH when you create the technology type; see Creating a Technology Type.
- **Source Code Type RC**. Enter the new file type you added to the CDR_FILE_TYPES lookup in the format $FILETYPES$*NEW_FILE_TYPE*.
- **Binary Flag RC**
- **Use Libname RC**
- **Position**
- **Allowed as Primary Flag RC**
- **Instantiated from Different Program**
- **Default Extension**. Enter the extension of the new file type, including the dot (.) preceding it; for example, `.xml`.

Planning Navigation in the External System to Business Areas

You can use the view CDR_PUB_GENERIC_BA_V to make the Oracle Life Sciences Data Hub object hierarchy—from Domains to Application Areas to Work Areas to Business Areas—to which the current user has access, available in the external system.

You can then use it to create a display in the external system's user interface that helps users find what they need; see Display User's Business Area Instances.

The user must log in using the database account.

> **Note:**
>
> This works only if you give the objects in the hierarchy meaningful names.

Editing cdrconfig.xml

When the user launches the IDE from Oracle Life Sciences Data Hub, the Oracle Life Sciences Hub client, cdrclient, generates a run script for the IDE launch based on what is defined in cdrconfig.xml.

Download the cdrconfig.xml file shipped with Oracle Life Sciences Hub and add information for your system's technology type to cdrconfig.xml as follows. See the *Oracle Life Sciences Data Hub Installation Guide* chapter on installing Oracle Life Sciences Hub, section on installing the client plug-in.

```
<TECHTYPE TYPE="$TECHTYPES$<tech_type_name>" NAME="<name>" VERSION="<version>">
<PARAM NAME="EXEHOME" VALUE="<complete_path_to_exe_file>"/>
<PARAM NAME="PREPROC" VALUE=""/>
<PARAM NAME="POSTPROC" VALUE=""/>
    <CMDLINEARGS>
<ARG POSITION="_POS1_" VALUE="_COMP999_"/>
</CMDLINEARGS>
</TECHTYPE>
```

For PREPROC, enter the full path to a batch file or shell script you have written, if any, that must be run immediately before running the .exe file for the IDE. For example, the OBIEE adapter needs to unzip the RPD file at this point and has a preprocessing .bat file for this purpose.

Use POSTPROC similarly, for a batch file or shell script that must be run immediately following the IDE launch.

CMDLINEARGS are for command line arguments to pass to the IDE executable.

You can specify additional argument values by adding argument/value pairs for positions 1 to 999 at the line:

```
<ARG POSITION="_POS1_" VALUE="_COMP999_"/>
```

For example, if you need to download a file to the IDE, add another argument/value pair such as `"_comp002_", "<filename>"` where <filename> is the file to be downloaded. The OBIEE adapter creates a default RPD file and downloads it to the IDE, where the user can edit it in the BI administrator's tool.

You must match argument values in the line above to the names you use in the Build_IDE_Cfg_Function, and output the argument/value pairs in the col_comp_list attribute of the po_Launch_Data parameter of the Build_IDE_Config_Function; see col_comp_list.

For example:

```
<TECHTYPE TYPE="$TECHTYPES$OBIEEDEV" NAME="AdminTool" VERSION="10.1.3.4">
<PARAM NAME="EXEHOME" VALUE="D:\OracleBI\server\Bin\AdminTool.exe"/>
<PARAM NAME="PREPROC" VALUE="unzipRPD.bat"/>
<PARAM NAME="POSTPROC" VALUE=""/>
    <CMDLINEARGS>
<ARG POSITION="_POS1_" VALUE="_COMP999_"/>
</CMDLINEARGS>
</TECHTYPE>
```

Creating Subdirectories on IDE Computers

For adapters that produce outputs you must create subdirectories on the Definer's PC to hold IDE job outputs (for example, any outputs that are defined as Planned Outputs of Programs) or any other files, such as error or log files.

Oracle Life Sciences Data Hub creates a directory structure on the Definer's PC that uses the organizational structure defined for the Program or Business Area in Oracle Life Sciences Hub. By default the location is: `%USERPROFILE%\Application Data\CDR`

`\cdrwork\`*`user_name`*`\`*`Domain(s)`*`\`*`Application_Area`*`\`*`Work_Area`*`\`*`launched_object_name`*`\`*`version`*. Oracle Life Sciences Hub creates the subdirectory you specify inside the *`version`* directory.

In the Build_IDE_Cfg_Function specify one or more subdirectory names in the col_sub_dirs attribute of the parameter po_LaunchData.

From the Install Guide: On each Definer's personal computer, load the CD-ROM that contains the unzipped files. InstallShield automatically runs setup.exe, which loads cdrconfig.xml and cdrclient.exe to a location the Definer specifies on his or her local computer. The default location is ProgramFilesDir\oracle\cdr where ProgramFilesDir is the registry entry for the value name ProgramFilesDir. If the specified location does not exist, the InstallShield creates it.

In addition, it sets the location for the CDR Work directory. By default this location is **%USERPROFILE%/Application Data/CDR**. Oracle recommends using this setting if you are installing the client IDE on a server for access by multiple users. It creates a separate work space for each user so they do not overwrite each other's files.

3

Developing an Adapter

Developing an adapter requires you to understand the overall process. You should plan your programs and packages carefully.

For details on this process, see the following:

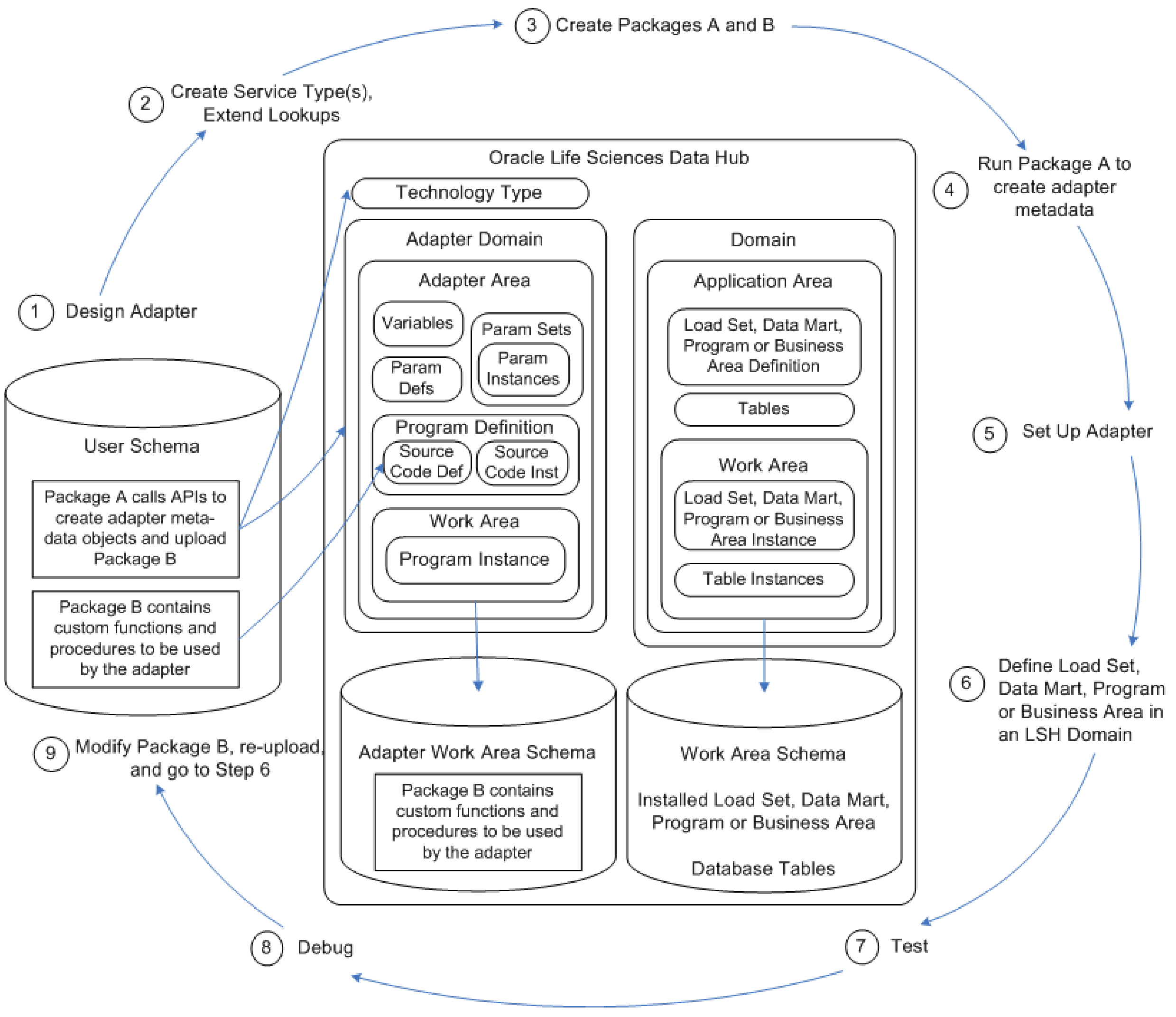
- Adapter Development Process
  The adapter development process is detailed. You should understand this process before beginning your development.
- Planning Programs and Packages
  Each custom function and procedure required for your adapter must be contained in a PL/SQL package (Package B) and the package must be uploaded to a Source Code definition in the Program definition in your Adapter Area.
- Using the Security API Package During Development
  You must call a security API, CDR_PUB_API_INITIALIZATION, from every package that calls an Oracle Life Sciences Data Hub API and that you intend to run from outside Oracle Life Sciences Hub.

Adapter Development Process

The adapter development process is detailed. You should understand this process before beginning your development.

The following diagram depicts the process required to create an adapter.

Figure 3-1 Adapter Development Process


1 Design Adapter
2 Create Service Type(s), Extend Lookups
3 Create Packages A and B
4 Run Package A to create adapter metadata
5 Set Up Adapter
6 Define Load Set, Data Mart, Program or Business Area in an LSH Domain
7 Test
8 Debug
9 Modify Package B, re-upload, and go to Step 6
User Schema
Package A calls APIs to create adapter meta-data objects and upload Package B
Package B contains custom functions and procedures to be used by the adapter
Oracle Life Sciences Data Hub
Technology Type
Adapter Domain
Adapter Area
Variables
Param Sets
Param Defs
Param Instances
Program Definition
Source Code Def
Source Code Inst
Work Area
Program Instance
Domain
Application Area
Load Set, Data Mart, Program or Business Area Definition
Tables
Work Area
Load Set, Data Mart, Program or Business Area Instance
Table Instances
Adapter Work Area Schema
Package B contains custom functions and procedures to be used by the adapter
Work Area Schema
Installed Load Set, Data Mart, Program or Business Area
Database Tables

To develop an adapter, work in SQL Developer or a similar tool, using your own Oracle Life Sciences Data Hub user account that is linked to a database account with Execute privileges on the API security package CDR_PUB_API_INITIALIZATION.

1. Design your adapter, following instructions in this guide.
2. Create one or more service types and extend lookups as required; see Planning Services and Adding Lookup Values.
3. In your own schema, create two PL/SQL packages:
   - **Package A: API Calls to Create metadata Objects**. This package must call the APIs described in Using APIs to Create Required Metadata Objects. Package A must create an Adapter Domain, Adapter Area, technology type, and all the required Oracle Life Sciences Hub objects (metadata) including a Program, upload Package B as source code for the Program, assign a user group to the Adapter Area, and install the Adapter Work Area.

> **Note:**
>
> When you run the APIs to create a technology type and an Adapter Area, you must enter the names of all your custom functions and procedures as input parameter values, and set flags to indicate whether or not you are using any of the three possible Parameter Sets.
> Therefore you must determine which custom functions and procedures and Parameter Sets you need, and give the functions and procedures names, **before** you can complete Package A. See Designing an Adapter.

- **Package B: Custom Functions and Procedures**. This package contains the functions and procedures that you write to do the work of the adapter; see Planning PL/SQL Functions and Procedures.

4. Run Package A. The system creates the required metadata objects, including the technology type, and uploads Package B as the source code for the adapter.
5. Follow the steps in Checking In Objects and Setting Their Validation Status, and Setting Up an Adapter to set up your adapter in your Oracle Life Sciences Hub instance.
6. In Oracle Life Sciences Hub, create an object definition and instance of the type relevant for your adapter: a Load Set, Data Mart, Program, or Business Area.

   You may need to create an Oracle Life Sciences Hub Domain, Application Area, and Work Area to contain the object (Load Set, Program, Data Mart, or Business Area) definition and instance; Table definitions and instances for the object to read from or write to; install the Work Area, and load data.
7. Test the adapter by checking that the required UI elements appear on screen and that it is possible to define, install and run the object.
8. You can debug and modify your functions and procedures in your own schema. Use APIs to check out the Program in the Adapter Area, upload the revised source code, and reinstall the Adapter Area's Work Area, and then test again. See the *Oracle Life Sciences Data Hub Application Programming Interface Guide*.
9. Modify Package B as required. Upload the modified package using the Modify Source Code public API.
10. Set the validation status of your objects to Quality Control and then to Production according to your standard operating procedures; see Checking In Objects and Setting Their Validation Status.

Planning Programs and Packages

Each custom function and procedure required for your adapter must be contained in a PL/SQL package (Package B) and the package must be uploaded to a Source Code definition in the Program definition in your Adapter Area.

The required functions and procedures can all be in the same package or in different packages. If multiple people are developing the adapter, it may be helpful for each of them to work on a separate package, upload each package to a different Source Code definition and create a different Program for each Source Code definition so that each

Program can be installed separately. However, all Program instances must be contained in the same Work Area.

> **Note:**
>
> It is very important to have only one Work Area in an Adapter Area. This ensures that all the custom PL/SQL packages you write are installed in the same schema.

Using the Security API Package During Development

You must call a security API, CDR_PUB_API_INITIALIZATION, from every package that calls an Oracle Life Sciences Data Hub API and that you intend to run from outside Oracle Life Sciences Hub.

This API contains three functions:

- **EnableApis** sets the EnableAPIs flag to True.
- **DisableApis** sets the EnableAPIs flag to False.
- **AreApisEnabled** returns the flag setting.

The initialization of almost every API calls the AreApisEnabled function of the security API, CDR_PUB_API_INITIALIZATION, to check if the EnableAPIs flag is set to True in the calling program. If EnableAPIs set to False, the initialization fails.

To set the EnableApis flag to True, your user account must have Execute privileges on the CDR_PUB_API_INITIALIZATION API.

1. Begin the body with the following code to call the function to enable APIs:

   ```
   call cdr_pub_api_initialization.enableApis (arguments);
   ```

   The arguments are described in cdr_pub_api_initialization itself.
2. At the end of the body, disable APIs to force the security check on the schema the next time the program is run:

   ```
   cdr_pub_api_intialization.disableApis (arguments);
   ```

Package A: Package A is always run outside of Oracle Life Sciences Hub. Therefore you need to call EnableApis at the beginning of your package body and call DisableAPIs at the end.

Package B: You develop Package B outside of Oracle Life Sciences Hub, so you must call EnableApis at the beginning of the package body and call DisableAPIs at the end. However, you upload Package B to a Program inside an Oracle Life Sciences Hub Adapter Area and the adapter calls the functions and procedures and runs them inside Oracle Life Sciences Hub. Packages that call CDR_PUB_API_INITIALIZATION from within Oracle LSH fail.

Therefore, if you call public APIs from Package B, you must call EnableApis and DisableApis so that you can run the package outside Oracle Life Sciences Hub, but when you are ready to upload the code to Oracle Life Sciences Hub, you must comment out the calls.

You may well want to use public APIs in your custom functions and procedures. For example, if you allow users to upload Table Columns during Load Set definition, you can take advantage of the public API for creating Table Descriptors as part of the upload_table_columns function.

4

Using APIs to Create Required Metadata Objects

Adapter Domains and the objects they contain are not accessible through the user interface. There is a public PL/SQL API for creating each of the objects required by an adapter.

Oracle Life Sciences Data Hub stores all the program code, parameters, and tables required for an adapter as defined objects within an Adapter Area which itself is contained in an Adapter Domain. Figure 1-1 shows the defined objects.

Additional information on APIs is available in the *Oracle Life Sciences Data Hub Application Programming Interface Guide*.

Call these APIs in the following order:

- Retrieving IDs
  Many APIs require IDs as input parameter values. You can retrieve these IDs from the database.
- Creating a Technology Type
  To create a technology type, you must use an API.
- Modifying a Technology Type
  If you need to modify your Technology Type, use the public API CDR_PUB_ATK_ADAPTER. MODIFYTECHTYPE.
- Creating an Adapter Domain
  An Adapter Domain is a container object that holds all the definitional objects required for an adapter.
- Modifying an Adapter Domain
  If you need to modify your Adapter Domain, use the public API CDR_PUB_ATK_ADAPTER. MODIFYADAPTERDOMAIN.
- Creating an Adapter Area
  An Adapter Area is a container object that can hold all the definitional objects required for an adapter.
- Modifying an Adapter Area
  If you need to modify your Adapter Area, use public API CDR_PUB_ATK_ADAPTER.MODIFYADAPTERAREA.
- Assigning a User Group to the Adapter Area
  In order to create the rest of the required objects, you must belong to a user group that is assigned to the Adapter Area, and you must have a role within the user group that allows you to create and modify each of the required objects.
- Creating a Work Area
  A Work Area contains instances of all the object definitions required for the adapter.

- Creating a Program Definition and Instance
  You must create a Program definition in the Adapter Area to store the source code required for the adapter. You must create an instance of the Program definition in the Work Area.
- Creating a Source Code Definition and Instance
  You must create at least one Source Code object definition inside the Program definition to contain the PL/SQL packages you write.
- Creating a Variable
  If you need user input to create a Load Set, Data Mart, Program, or Business Area —for example, an Oracle Remote Location name and connection—you must define a Parameter to collect the information. For each Parameter, you must define a Variable for the Parameter to reference.
- Creating a Parameter
  Use Parameters to allow users to enter values during the definition or execution of a Load Set, Data Mart, Program, or Business Area. You must create a defined Parameter object for each parameter required and handle the user input in your source code.
- Creating a Parameter Set
  You may create one, two, or three Parameter Sets to collect user input during the definition, installation, or execution of a Load Set, Data Mart, or Program, or the launch of a visualization tool.
- Creating a Parameter Instance in a Parameter Set
  For each Parameter definition you have created, create a Parameter instance in the appropriate Parameter Set.
- Creating a Table Definition
  If you are creating a Load Set adapter and the external source data system has fixed data structures, you may want to define those data structures as Tables in the Adapter Area rather than uploading them or forcing the user to create them manually each time a user creates a Load Set.
- Installing the Work Area
  After you have created the Adapter Domain and all the definitional objects within it, install the Work Area by calling the API CDR_PUB_DF_WORKAREA.INSTALLWACONTROLLER and the Program instance it contains.

Retrieving IDs

Many APIs require IDs as input parameter values. You can retrieve these IDs from the database.

See the following for details:

- Getting Your Company ID
- Getting an Object's Prref_Id and Prref_Ver

Getting Your Company ID

The company ID is part of the primary key for every object.

To get your company ID, run the API CDR_PUB_DEF_FACTORY_UTILS.GETCOMPANYID.

Save the value for repeated use or create a local variable for it.

Getting an Object's Prref_Id and Prref_Ver

Use the following query to retrieve these values:

```
select prref_id, prref_ver from cdr_program_refs_v
where COMPANY_ID = <your_company_id> and
WA_OBJ_ID = <the_adapter_work_area_id> and
MASTER_PRREF_OBJ_ID = <the_obj_id_of_the_object> and
MASTER_PRREF_OBJ_VER = <the_obj_ver_of_the_object>;
```

The primary purpose of the Prref ID is to provide an execution context for executable objects contained in complex objects, such as Program instances contained in Report Set or Workflow instances. In these cases, the Master Prref Obj ID is the Object ID of the Report Set or Workflow instance that owns the Program instance.

For consistency, all object instances must have a Prref ID. In the case of object instances located directly in a Work Area, the object's Master Prref Obj ID is the same as its Object ID. For example, the Master Prref ID of a Program instance contained in a Work Area is the same as its Object ID; so if you know its Object ID, you know its Prref ID.

Note:

When you create an object using an API, save its Object ID, which is an output parameter value. You may need the ID to get the object's Prref ID or to create child objects.

Creating a Technology Type

To create a technology type, you must use an API.

See the following for details:

- Call the Create Technology Type API
- Sample Technology Type Settings

Call the Create Technology Type API

To create an Technology Type, call the public API CDR_PUB_ATK_ADAPTER.POPULATETECHTYPES.

Its signature is:

```
PROCEDURE populateTechTypes (
p_api_version IN NUMBER
,p_init_msg_list IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_commit IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_validation_level IN NUMBER default CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL
,x_return_status OUT NOCOPY VARCHAR2
,x_msg_count OUT NOCOPY NUMBER
,x_msg_data OUT NOCOPY VARCHAR2
, pio_techTypeRow IN OUT NOCOPY cdr_tech_types%rowtype
);
```

param pio_techTypeRow is a mandatory parameter of row type cdr_tech_types table that contains object attributes. Enter values as follows:

- **tech_type_id**. Contact Oracle Support to get a 12-digit ID for your technology type that is unique across all adapters developed for use with Oracle Life Sciences Data Hub.

  > **Note:**
  >
  > Make a note of the number you enter; you will need it when you create an Adapter Area.

- **tech_name_rc**. Enter a name for the new technology type you are creating. You must add the name to the CDR_TECH_TYPES lookup (see Adding Lookup Values).
- **tech version**. Enter the correct version of the external system.
- **service_type_rc**. The service type required for this technology type. You must add this service type value to the CDR_SERVICE_TYPES lookup (see Adding Lookup Values).
- **build_ide_cfg_function**. (Visualization and Program adapters only.) Enter the name of your Pre-installation function, if any; see Build_IDE_Cfg_Function.

  > **Note:**
  >
  > For this function name and the others below, do not include the schema name. Enter only *`package_name.procedure_or_function_name`*

- **owb_operator**. Enter the Oracle Warehouse Builder operator that Oracle LSH should use for this adapter:
  - If the external system is located on the database, enter: `CdrPLSQLImmediate_1` These adapters use PL/SQL to execute their defined objects.
  - If the external system is located on an operating system, enter: `CdrSERVICE_1`. These adapters use the Distributed Processing (DP) Server to execute their defined objects; see the chapter on setting up services in the *Oracle LSH System Administrator's Guide* for more information.
  - If your adapter is for an IDE, enter: `CdrService`
- **program_type_rc**. Enter the type of defined object created using this adapter: `$PROGRAMTYPES$LOADSET`, `$PROGRAMTYPES$DATA_MART`, `$PROGRAMTYPES$PROGRAM`or `$PROGRAMTYPES$BUSAREA`.
- **pre_install_function**. Enter the name of your Pre-installation function, if any; see Pre_Install_Function.
- **install_function**. Enter the name of your Installation function, if any; see Install_Function.
- **post_install_function**. Enter the name of your Postinstallation function, if any; see Post_Install_Function.

- **pre_execution_function**. Enter the name of your Pre-execution function, if any; see Pre_Execution_Function.
- **execution_function**. Enter the name of your Execution function, if any; see Execution_Function.
- **post_execution_function**. Enter the name of your Postexecution function, if any; see Post_Execution_Function.

Note:

If any of these functions returns an error the job returns an error. If any of these programs returns a warning then the job returns a warning unless another part of the job generated an error.

Sample Technology Type Settings

The following table shows the values used by a sampling of the adapters shipped with Oracle Life Sciences Data Hub.

You can see all settings for the shipped adapters in CDR_TECH_TYPES_V.

Table 4-1 Technology Types

Column	Oracle Tables and Views Load Set Adapter	Text Load Set Adapter	SAS Load Set Adapter	SAS Data Mart Adapter	Generic Visualization Business Area Adapter
TECH_NAME_RC	$TECHTYPES$ORACLE	$TECHTYPES$TEXT	$TECHTYPES$SASLOADSET	$TECHTYPES$SASDATAMART	$TECHTYPES$GVA
TECH_VERSION	9.x	1	8.x	6.12	1.0.0.1
SERVICE_TYPE_RC	$SERVICETYPES$PLSQL	$SERVICETYPES$TEXT	$SERVICETYPES$SAS8	$SERVICETYPES$SAS8	Null
BUILD_IDE_CFG_FUNCTION	Null	Null	Null	Null	Null
BUILD_EXE_CFG_FUNCTION	Null	Null	Null	Null	Null
OWB_OPERATOR	CdrPLSQLImmediate_1	CdrSERVICE_1	CdrSERVICE_1	CdrSERVICE_1	CdrSERVICE_1
PROGRAM_TYPE_RC	$PROGRAMTYPES$LOADSET	$PROGRAMTYPES$LOADSET	$PROGRAMTYPES$LOADSET	$PROGRAMTYPES$DATA_MART	$PROGRAMTYPES$BUSAREA
PRE_INSTALL_FUNCTION	Null	Null	Null	Null	Null
INSTALL_FUNCTION	Null	Null	Null	Null	CDR_GV_ADAPTER.INSTALL
POST_INSTALL_FUNCTION	Null	Null	Null	Null	Null
PRE_EXECUTION_FUNCTION	Null	CDR_ATK_Text_Services.ExePreProcessor	CDR_EXE_SAS.buildSasConfigTmpLS	CDR_ATK_DM_SAS_SERVICES.executeDatamart	Null

Table 4-1 (Cont.) Technology Types

Column	Oracle Tables and Views Load Set Adapter	Text Load Set Adapter	SAS Load Set Adapter	SAS Data Mart Adapter	Generic Visualization Business Area Adapter
EXECUTION_FUNCTION	CDR_ATK_oracledb.loadsetprocessing	Null	Null	Null	Null
POST_EXECUTION_FUNCTION	Null	Null	Null	Null	Null

Modifying a Technology Type

If you need to modify your Technology Type, use the public API CDR_PUB_ATK_ADAPTER. MODIFYTECHTYPE.

Its signature is:

```
Package cdr_pub_atk_adapter
PROCEDURE modifyTechType(p_api_version IN NUMBER,p_init_msg_list IN VARCHAR2 default
CDR_PUB_DEF_CONSTANTS.G_FALSE,p_commit IN VARCHAR2 default
CDR_PUB_DEF_CONSTANTS.G_FALSE,p_validation_level IN NUMBER default
CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,x_return_status OUT NOCOPY
VARCHAR2,x_msg_count OUT NOCOPY NUMBER,x_msg_data OUT NOCOPY VARCHAR2,
pio_techTypeRow IN OUT NOCOPY cdr_tech_types%rowtype);
```

See the description of parameter pio_techTypeRow in Creating a Technology Type.

Creating an Adapter Domain

An Adapter Domain is a container object that holds all the definitional objects required for an adapter.

Note:

Creating an Adapter Domain and Adapter Area is required for Load Set, Data Mart, and Visualization adapters, but may not be required for IDE adapters if the IDE requires the user to log in.

See the following for details:

- Call the Create Adapter Domain API
- Save the Adapter Domain ID for Future Use

Call the Create Adapter Domain API

To create an Adapter Domain, call the public API CDR_PUB_ATK_ADAPTER.CREATEADAPTERDOMAIN. Its signature is:

```
PROCEDURE createAdapterDomain(
p_api_version IN NUMBER
,p_init_msg_list IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_commit IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_validation_level IN NUMBER default CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL
,x_return_status OUT NOCOPY VARCHAR2
,x_msg_count OUT NOCOPY NUMBER
,x_msg_data OUT NOCOPY VARCHAR2
, pio_adapterDomainNaming IN OUT NOCOPY cdr_naming_version_obj_type
) ;
```

It has one mandatory parameter, PIO_ADAPTERDOMAINNAMING, of table type CDR_NAMING_VERSION_OBJ_TYPE. Enter values as follows:

- **company_id** = Enter your company ID; see Getting Your Company ID.
- **obj_id** = null
- **obj_ver** = null
- **object_type_rc** = `'$OBJTYPES$ADAPTERDOMAIN'`
- **name** = Enter a name for the Adapter Domain.
- **namespace_obj_id** = null
- **namespace_obj_ver** = null
- **namespace_start_obj_ver** = null
- **namespace_end_obj_ver** = `cdr_def_constants.cdr_max_def_object_version`
- **owning_location_rc** = null
- **checked_out_flag_rc** = null
- **checked_out_id** = null
- **object_subtype_id** = null
- **description** = Enter a description of the Adapter Domain.
- **copied_from_company_id** = null
- **copied_from_obj_id** = null
- **copied_from_obj_ver** = null
- **ref_company_id** = null
- **ref_obj_ver** = null
- **object_version_number** = 1
- **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
- **validation_status_rc** = null
- **version_label** = null

Save the Adapter Domain ID for Future Use

You will need the internal ID for this Adapter Domain when you create the Adapter Area.

Modifying an Adapter Domain

If you need to modify your Adapter Domain, use the public API CDR_PUB_ATK_ADAPTER. MODIFYADAPTERDOMAIN.

Its signature is:

```
Package cdr_pub_atk_adapter

PROCEDURE modifyAdapterDomain(
p_api_version IN NUMBER
,p_init_msg_list IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_commit IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_validation_level IN NUMBER default CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL
,x_return_status OUT NOCOPY VARCHAR2
,x_msg_count OUT NOCOPY NUMBER
,x_msg_data OUT NOCOPY VARCHAR2
, pio_adapterDomainNaming IN OUT NOCOPY cdr_naming_version_obj_type
) ;
```

See the description of parameter pio_adapterDomainNaming in Creating an Adapter Domain.

Creating an Adapter Area

An Adapter Area is a container object that can hold all the definitional objects required for an adapter.

It is similar in function to an Oracle Life Sciences Data Hub Application Area, which is described in the *Oracle LSH Implementation Guide*.

Normally you need only one Adapter Area in an Adapter Domain. However, if you are developing more than one adapter for a single external system, (like the multiple shipped Oracle Clinical adapters) you should create one Adapter Area for each adapter.

For details, see the following:

- Call the Create Adapter Area API
- Save the Adapter Area ID for Future Use
- Sample Adapter Settings

Call the Create Adapter Area API

You must create a new Adapter Area using the public API CDR_PUB_ATK_ADAPTER.CREATEADAPTERAREA.

Its signature is:

```
PROCEDURE createAdapterArea(
p_api_version IN NUMBER
,p_init_msg_list IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_commit IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_validation_level IN NUMBER default CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL
,x_return_status OUT NOCOPY VARCHAR2
,x_msg_count OUT NOCOPY NUMBER
,x_msg_data OUT NOCOPY VARCHAR2
, pio_adapterAreaNaming IN OUT NOCOPY cdr_naming_version_obj_type
, pio_adapterAreaRow IN OUT NOCOPY cdr_adapter_areas%rowtype
) ;
```

Enter parameter values as follows:

> **Note:**
>
> When you supply the name of a function or procedure you have written, do not include the schema name. Enter only `*package_name.procedure_or_function_name*`

- **PIO_ADAPTERAREANAMING**. This is a mandatory parameter of table type CDR_NAMING_VERSION_OBJ_TYPE. Enter attribute values as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null

    > **Note:**
    >
    > The system generates this ID. Save the output parameter value. You will need it to create objects inside the Adapter Area.

  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$ADAPTERAREA'`
  - **name** = Enter a name for the Adapter Area. The name appears in the UI in the Adapter drop-down list on the object's Create page.
  - **namespace_obj_id** = *`Your_Adapter_Domain_ID`*
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - **owning_location_rc** = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null
  - **description** = '*`Enter_a_Description_for_the_Program`*'
  - **copied_from_company_id** = null
  - **copied_from_obj_id** = null
  - **copied_from_obj_ver** = null
  - **ref_company_id** = null
  - **ref_obj_ver** = null

  - **object_version_number** = 1
  - **status_rc** = null
  - **validation_status_rc** = null
  - **version_label** = null
- **PIO_ADAPTERAREAROW**. This is a parameter of row type CDR_ADAPTER_AREAS table. Enter object attribute values as follows:

> **Note:**
>
> Adding a row to the Adapter Areas table is required for Load Set, Data Mart, and Visualization adapters, but not for IDE adapters.

  - **company_id**. *Enter_your_company_ID*
  - **obj_id**. Set to Null. The system enters the value.
  - **obj_ver**. Enter 1. Or, if you are updating the adapter, increment the version number by 1.
  - **adapter_name**. Enter a name for your adapter.
  - **adapter_version**. Enter the version number of the external system. This is for your information only; this field has no effect.
  - **adapter_type**. Enter one of the following adapter types:
    - LOADSET for Load Set adapters
    - DATAMART for Data Mart adapters
    - BUSAREA for Business Area adapters
    - PROGRAM for Program adapters
  - **tech_type_id**. Enter the tech type ID for your adapter.
  - **allow_column_upload**. (Applies only to Load Set adapters.) Enter YES, NO, or FILE. If set to **NO**, a user defining a Load Set using this adapter cannot upload column data structures from an external system. If you want the user to be able to upload these data structures, enter YES if the source system is a database and FILE if the source system stores data in files, such as SAS data sets or xml files.
    If set to **YES** or **FILE**, the Upload Column button appears in the Table Descriptor properties page. Set to **NO** if your adapter is not a Load Set-type adapter or if it is a Load Set adapter but the table structure is predefined, as in some of the Oracle Clinical adapters; see Planning Data Structures.

    The following shipped Load Set adapters use a column upload function: Oracle Clinical Data Extract Oracle Views, Oracle Clinical Data Extract SAS Views, Oracle Tables and Views, and SAS.
  - **column_upload_function**. Applies only to Load Set adapters. If the Allow_Column_Upload flag is set to YES or FILE, write a function to upload data structures and enter the *schema_id.package_name* of the function.
  - **allow_manual_tab_desc_flag**. Enter YES or NO. Set to YES to enable the Add button in the Table Descriptors subtab in user interface, so that users can manually define Table Descriptors. Set to NO to render the button inactive.

- **auto_add_tab_desc_lov**. Applies only to Load Set adapters. See Object Definition Functions and Procedures for information.
- **allow_auto_add_tab_desc**. Enter YES or NO. If set to NO, the Definer will not be able to choose from a list of Tables to create Table Descriptors. If set to YES, you must write a program to create a list of appropriate Tables and enter its name in Auto_Add_Tab_Desc_ Function. You must also create a program to create the selected Table Descriptors and enter its name in Auto_Add_Tab_Desc_LOV.
- **auto_add_tab_desc_function**. Applies only to Load Set adapters. See Object Definition Functions and Procedures for information.
- **currency_function**. See Object Definition Functions and Procedures for information.
- **define_time_function**. See Object Definition Functions and Procedures for information.
- **define_time_connect_flag**. Enter YES if, in order to define an object through this adapter, it is necessary to connect to a remote database. This is the case for Oracle-based Load Set adapters, for example, to get a list of tables on a remote database.
  If you enter YES, you must create a Parameter instance in the define-time Parameter Set to collect the remote location information; see Table Descriptor Define-Time Parameters. You must then use the Parameter value(s) in your code; for example, in the Auto_Add_Tab_Desc_Function to return the list of tables from the remote system for the user to select.

  Enter NO if no remote connection is required during object definition.
- **install_time_connect_flag**. Enter YES if, in order to install an object created through this adapter, it is necessary to connect to a remote database. This is the case for Orace-based Load Set adapters, for example, if the Definer chooses to map Table Descriptors to Table instances defined as views. If set to YES, you must create a Parameter instance in the define-time Parameter Set to collect the remote location information; see Table Descriptor Define-Time Parameters.
  Enter NO if no remote connection is required during object installation.
- **runtime_connect_flag**. Enter YES if, in order to run an object created through this adapter, it is necessary to connect to a remote database. This is the case for SAS and Text Load Set adapters that load a file. If set to YES, you must create a Parameter in the runtime Parameter Set to collect a value for the remote location; see Object Execution Functions and Procedures.
  Enter NO if no remote connection is required to run the object.
- **tables_as_views_flag**. (Applies only to Load Set adapters.) Enter YES to allow the Definer to create the Load Set's target Table instances as passthrough views to tables in the external system, so that the user can view data in the external system. Entering YES here adds the item "Create Table as a View" from the Process Type drop-down in the properties page of Table instances mapped to target Table Descriptors.
  Enter NO if the external system cannot support this functionality or if you do not want to use it.

  The following shipped adapters have this flag set to YES: Oracle Clinical Labs, Oracle Clinical Data Extract Oracle Views, and Oracle Tables and Views.

– **tables_as_views_function**. Set to Null. This function is not currently used, even by Load Sets that support tables as views.

– **active_flag_rc**. Enter `'$YESNO$YES'`

> **Note:**
>
> You can always set this flag to `YES`. The adapter will not actually become available for use until you have assigned user groups to the Adapter Area.

– **def_param_flag_rc**. Enter `'$YESNO$YES'` if this adapter has a Parameter Set called PARAMETERSET_LOADSETLEVEL_DEF. See Object Definition Functions and Procedures for further information. The system then displays the Parameters in this Parameter Set as define-time attributes of Load Sets, Data Marts, Program, or Business Areas created with this adapter.
Enter `'$YESNO$NO'` if this adapter does not have a Parameter Set called PARAMETERSET_LOADSETLEVEL_DEF.

– **run_param_flag_rc**. Enter `'$YESNO$YES'` if this adapter has a Parameter Set called PARAMETERSET_LOADSETLEVEL_RUN. See Object Execution Functions and Procedures for further information. The system then displays the Parameters in this Parameter Set as runtime Parameters in the Parameters subtab of Load Sets, Data Marts, Program, or Business Areas and in the Execution Setup of Load Sets or Data Marts created with this adapter.
Enter `'$YESNO$NO'` if this adapter does not have a Parameter Set called PARAMETERSET_LOADSETLEVEL_RUN.

– **td_param_flag_rc**. (Applies only to Load Set adapters.) Enter `'$YESNO$YES'` if this adapter has a Parameter Set called PARAMETERSET_OPERATORLEVEL. See Table Descriptor Define-Time Parameters for further information. The system then displays the Parameters in this Parameter Set as define-time attributes of Load Sets created with this adapter.
Enter `'$YESNO$NO'` if this adapter does not have a Parameter Set called PARAMETERSET_OPERATORLEVEL

– **status_recalc_function**. See Object Definition Functions and Procedures for information.

– **security_recalc_flag_rc**. Enter `'$YESNO$YES'` if you write a Security Recalulation function to synchronize Oracle Life Sciences Data Hub security with the security of the external system; for example, for a Business Area adapter.
Enter `'$YESNO$NO'` if there is no security synchronization between Oracle Life Sciences Hub and the external system. See Synchronizing Security with Integrated Environments.

– **security_recalc_function**. See Object Execution Functions and Procedures for information.

– **install_schema_flag_rc**. If your adapter requires a dedicated Oracle database schema, enter `'$YESNO$YES'`. If not, enter `'$YESNO$NO'`.
If you set this flag to `'$YESNO$YES'`, Oracle Life Sciences Hub creates an additional schema dedicated to this adapter when you install the Work Area containing the Business Area or other adapter-related object, and gives the

additional schema the same name as the Work Area schema plus the suffix you specify for **install_schema_suffix**. You can use the install_functions to populate this schema with whatever objects your adapter requires. Oracle Life Sciences Hub reinstalls the schema each time the Work Area is installed, using the same installation type specified for the Work Area. No more than one additional schema per adapter type is created per Work Area.

– **install_schema_suffix**. If you set the **install_schema_flag_rc** to `'$YESNO$YES'`, enter text. The system appends this text to the Work Area schema name to create the name for the additional schema.

Save the Adapter Area ID for Future Use

The input/output Parameter PIO_SOURCECDRNAMING returns the Object ID (`obj_id`) of the Adapter Area.

You will need this ID each time you create an object in the Adapter Area.

Sample Adapter Settings

The following table shows the Adapter Area settings used by some of the adapters shipped with Oracle Life Sciences Data Hub.

You can see all settings for the shipped adapters in CDR_ADAPTER_AREAS_V.

Note:

The system treats No and Null the same way.

Table 4-2 Sample Adapter Settings

Column Name	Oracle Tables and Views Load Set Adapter	Text Load Set Adapter	SAS Load Set Adapter	SAS Data Mart Adapter	Generic Visualization Business Area Adapter
OBJ_VER	1	1	1	1	1
ADAPTER_NAME	ORACLE_DATABASE	Text	SAS	SAS EXPORT	GENERIC_VISUALIZATION
ADAPTER_VERSION	1	1	8.2	1	1.0.1.2
ADAPTER_TYPE	LOADSET	LOADSET	LOADSET	DATAMART	BUSAREA
TECHNOLOGY_TYPE_ID	(Generated)	(Generated)	(Generated)	(Generated)	(Generated)
ALLOW_COLUMN_UPLOAD	`YES`	`NO`	File	`NO`	`NO`
COLUMN_UPLOAD_FUNCTION	CDR_ATK_OracleDB_Services.UploadColumns	Null	CDR_ATK_SAS_Services.UploadOperatorColumns	`NO`	Null
ALLOW_MANUAL_TAB_DESC_FLAG	`YES`	`YES`	`YES`	`YES`	`NO`

Table 4-2 (Cont.) Sample Adapter Settings

Column Name	Oracle Tables and Views Load Set Adapter	Text Load Set Adapter	SAS Load Set Adapter	SAS Data Mart Adapter	Generic Visualization Business Area Adapter
AUTO_ADD_TAB_DESC_LOV	CDR_ATK_OracleDB_Services.CreateMultipleOperators	NO	NO	NO	NO
ALLOW_AUTO_ADD_TAB_DESC	YES	NO	NO	YES	NO
AUTO_ADD_TAB_DESC_FUNCTION	CDR_ATK_OracleDB_Services.GetDataOperList	Null	Null	NO	Null
CURRENCY_FUNCTION	Null	Null	Null	NO	Null
DEFINE_TIME_FUNCTION	CDR_ATK_OracleDB_Services.CreateLogFilePlannedOutput	CDR_ATK_Text_Services.CreateLogFilePlannedOutput	CDR_ATK_SAS_Services.CreateLogFilePlannedOutput	CDR_ATK_DM_SAS_SERVICES.createPlannedOutput	Null
DEFINE_TIME_CONNECT_FLAG	YES	NO	NO	NO	NO
INSTALL_TIME_CONNECT_FLAG	YES	NO	NO	NO	NO
RUNTIME_CONNECT_FLAG	YES	NO	NO	NO	NO
TABLES_AS_VIEWS_FLAG	YES	NO	NO	NO	NO
TABLES_AS_VIEWS_FUNCTION	Null	Null	Null	NO	Null
ACTIVE_FLAG_RC	$YESNO$YES	$YESNO$YES	$YESNO$YES	$YESNO$YES	$YESNO$YES
DEF_PARAM_FLAG_RC	$YESNO$YES	$YESNO$YES	$YESNO$YES	$YESNO$YES	$YESNO$YES
RUN_PARAM_FLAG_RC	$YESNO$YES	$YESNO$YES	$YESNO$YES	$YESNO$YES	$YESNO$NO
TD_PARAM_FLAG_RC	$YESNO$YES	$YESNO$NO	$YESNO$NO	$YESNO$NO	$YESNO$NO
STATUS_RECALC_FUNCTION	Null	Null	Null	CDR_ATK_DM_SAS_SERVICES.synchronizeDatamart	Null
SECURITY_RECALC_FLAG_RC	$YESNO$NO	$YESNO$NO	$YESNO$NO	$YESNO$NO	$YESNO$NO
SECURITY_RECALC_FUNC	Null	Null	Null	Null	Null
INSTALL_SCHEMA_FLAG_RC	$YESNO$YES	$YESNO$NO	$YESNO$NO	$YESNO$NO	$YESNO$NO

Table 4-2 (Cont.) Sample Adapter Settings

Column Name	Oracle Tables and Views Load Set Adapter	Text Load Set Adapter	SAS Load Set Adapter	SAS Data Mart Adapter	Generic Visualization Business Area Adapter
INSTALL_SCHEMA_SUFFIX	Null	Null	Null	Null	Null

Modifying an Adapter Area

If you need to modify your Adapter Area, use public API CDR_PUB_ATK_ADAPTER.MODIFYADAPTERAREA.

Its signature is:

```
Package cdr_pub_atk_adapter
PROCEDURE modifyAdapterArea(
p_api_version IN NUMBER
,p_init_msg_list IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_commit IN VARCHAR2 default CDR_PUB_DEF_CONSTANTS.G_FALSE
,p_validation_level IN NUMBER default CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL
,x_return_status OUT NOCOPY VARCHAR2
,x_msg_count OUT NOCOPY NUMBER
,x_msg_data OUT NOCOPY VARCHAR2
, pio_adapterAreaNaming IN OUT NOCOPY cdr_naming_version_obj_type
, pio_adapterAreaRow IN OUT NOCOPY cdr_adapter_areas%rowtype
) ;
```

See the description of the parameters in Creating an Adapter Area.

Assigning a User Group to the Adapter Area

In order to create the rest of the required objects, you must belong to a user group that is assigned to the Adapter Area, and you must have a role within the user group that allows you to create and modify each of the required objects.

Note:

Only the people who will build the adapter should have these privileges within a user group assigned to the Adapter Area. If the same people will later define Load Sets, Data Marts, Program, or Business Areas that use this adapter, you may want to remove them from this user group or ask them to always log in as a different user in a different user group so that they do not inadvertently modify a Parameter definition and invalidate the adapter. See Assigning User Groups to the Adapter Area.

You can assign user groups to Adapters in the user interface or by calling an API.

Note:

If you are creating an adapter to be used by other companies, use the API method to assign a user group. In the installation script for your adapter, create an input parameter to accept one or more user group IDs in the customer company.

For more details, see the following:

- User Interface Method
- API Method

User Interface Method

User interface instructions are included in the chapter on setting up adapters in the *Oracle LSH System Administrator's Guide*.

API Method

Call the API CDR_PUB_SECURITY_PKG.ASSIGNUSRGRPTOOBJ. Its signature is:

```
PROCEDURE ASSIGNUSRGRPTOOBJ(
  P_API_VERSION  IN    NUMBER,
  P_INIT_MSG_LIST  IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT  IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL  IN    NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS  OUT    VARCHAR2,
  X_MSG_COUNT  OUT    NUMBER,
  X_MSG_DATA  OUT    VARCHAR2,
  PI_BASEOBJECTTYPE  IN OUT    CDR_BASE_OBJ_TYPE,
  PI_CDROBJUGCOLL  IN    CDR_OBJ_UG_COLL
);
```

Enter values for the parameter PI_BASEOBJECTTYPE as follows:

- **company_id**. Set to null.
- **obj_id**. Enter the obj_id of the Adapter Area.
- **obj_ver**. Enter 1
- **object_version_number**. Set to null.
- **namespace_obj_id**. Enter the obj_id of the Adapter Domain
- **namespace_object_ver**. Enter 1

Enter values for the parameter PI_CDROBJUGCOLL as follows:

- **ug_company_id**. Set to null.
- **obj_company_id**. Set to null.
- **user_group_id**. Enter the user_group_id of the user group.
- **obj_id**. Enter the obj_id of the Adapter Area.
- **exclusion_flag**. Set to `'N'`.
- **object_version_number**. Set to null.

Creating a Work Area

A Work Area contains instances of all the object definitions required for the adapter.

Work Areas are described in the *Oracle LSH Implementation Guide*.

Note:

Save the Obj_ID of the Work Area for use when you create objects inside it.

For more information, see the following:

- Call the Create Work Area API
- Save the Work Area ID for Future Use

Call the Create Work Area API

To create a Work Area, call the API CDR_PUB_DF_WORKAREA.CREATEWORKAREA.

Its signature is:

```
PROCEDURE CREATEWORKAREA(
     P_API_VERSION             IN    NUMBER,
     P_INIT_MSG_LIST           IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_COMMIT                  IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_VALIDATION_LEVEL        IN    NUMBER :=
CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
     X_RETURN_STATUS           OUT    VARCHAR2,
     X_MSG_COUNT               OUT    NUMBER,
     X_MSG_DATA                OUT    VARCHAR2,
     PIO_SOURCECDRNAMING       IN OUT    CDR_NAMING_VERSION_OBJ_TYPE,
     PIO_WORKAREAOBJTYPE       IN OUT    CDR_WORKAREA_OBJ_TYPE,
     PI_DEFCLASSIFICATIONCOLL  IN    CDR_CLASSIFICATIONS_COLL
);
```

Enter Parameter values as follows:

- **PIO_SOURCECDRNAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values for the Work Area, as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$WORKAREA'`
  - **name** = '*`Enter_a_name_for_the_Work_Area`*'
  - **namespace_obj_id** = *`Your_Adapter_Area's_obj_id`*
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1

- **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
- **owning_location_rc** = null
- **checked_out_flag_rc** = `'$YESNO$NO'`
- **checked_out_id** = null
- **object_subtype_id** = null
- **description** = '*`Enter_a_Description_for_the_Work_Area`*'
- **copied_from_company_id** = null
- **copied_from_obj_id** = null
- **copied_from_obj_ver** = null
- **ref_company_id** = null
- **ref_obj_ver** = null
- **object_version_number** = 1
- **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
- validation_status_rc = null
- **version_label** = null

- **PI_WORKAREAOBJTYPE**. Enter CDR_WORKAREA_OBJ_TYPE values as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = 1
  - **label** = `'Standard'`
  - **workarea_status_rc** =null
  - **last_status_change_ts** = sysdate
  - **usage_intent_rc** = `'$SYSVALDNSTEPS$PRODUCTION'`

Note:

If you set the Usage Intent to Production, you can still modify your source code and defined objects as necessary and reinstall the Work Area. Then when you have finished developing the adapter, if you want to upgrade all objects' validation status to Production, you can do so. If you prefer, set Usage Intent to Development now and use the API CDR_PUB_DF_WORKAREA.UPDATEUSAGEINTENT to change it to Production later.
An Adapter Area can contain only one Work Area.

  - **cloned_from_company_id** = null
  - **cloned_from_obj_id** = null
  - **cloned_from_obj_ver** = null

  – **wa_runtime_status_rc** = null

Save the Work Area ID for Future Use

The input/output Parameter PIO_SOURCECDRNAMING returns the Object ID (**obj_id**) of the Work Area.

Creating a Program Definition and Instance

You must create a Program definition in the Adapter Area to store the source code required for the adapter. You must create an instance of the Program definition in the Work Area.

For details, see the following:

- Query for the Tech Type ID
- Call the Create Program API
- Save the Program Definition and Instance IDs for Future Use

Query for the Tech Type ID

Before you call the API, run the following query to retrieve the local tech type ID for the PL/SQL technology type:

```
select tech_type_id from cdr_tech_types where tech_name_rc='$TECHTYPES$PLSQL';
```

Note:

Save the tech type ID for use in creating Source Code definitions as well as the Program definition.

Call the Create Program API

To create a Program definition and an instance of it, call the API CDR_PUB_DF_PROGRAM.CREATEPROGRAM.

Its signature is:

```
PROCEDURE CREATEPROGRAM(
     P_API_VERSION              IN    NUMBER,
     P_INIT_MSG_LIST            IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_COMMIT                   IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_VALIDATION_LEVEL         IN    NUMBER :=
CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
     X_RETURN_STATUS            OUT    VARCHAR2,
     X_MSG_COUNT                OUT    NUMBER,
     X_MSG_DATA                 OUT    VARCHAR2,
     PIO_SOURCECDRNAMING        IN OUT CDR_NAMING_VERSION_OBJ_TYPE,
     PI_CDRPRGOBJTYPE           IN    CDR_PROGRAM_OBJ_TYPE,
     PI_CREATEOBJECT            IN    VARCHAR2,
     PI_INSTANCE_SUBTYPE_ID     IN    CDR_NAMINGS.OBJECT_SUBTYPE_ID%TYPE,
     PI_DEFCLASSIFICATIONCOLL   IN    CDR_CLASSIFICATIONS_COLL,
     PI_INSTCLASSIFICATIONCOLL  IN    CDR_CLASSIFICATIONS_COLL
);
```

Enter Parameter values as follows:

- **PIO_SOURCECDRNAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values that apply to the Program definition, as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = null
  - **name** = '`Enter_a_name_for_the_Program`'
  - **namespace_obj_id** = `Your_WorkArea's_obj_ID`
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - **owning_location_rc** = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null
  - **description** = '`Enter_a_Description_for_the_Program`'
  - **copied_from_company_id** = null
  - **copied_from_obj_id** = null
  - **copied_from_obj_ver** = null
  - **ref_company_id** = null
  - **ref_obj_ver** = null
  - **object_version_number** = 1
  - **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  - **validation_status_rc** = null
  - **version_label** = null
- **PI_CDRPRGOBJTYPE**. Enter CDR_PROGRAM_OBJ_TYPE values that apply to the Program definition, as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = 1
  - **tech_type_id** = `Enter_your_Tech_Type_ID`
  - **manual_validation_flag_rc** = `'$YESNO$NO'`
- **PI_CREATEOBJECT** = `BOTH`
- **PI_DEFCLASSIFICATIONCOLL** = null

- **PI_INSTCLASSIFICATIONCOLL** = null

Save the Program Definition and Instance IDs for Future Use

The input/output Parameter PIO_SOURCECDRNAMING returns the Object ID (`obj_id`) of the Program instance.

To get the Object ID of the Program definition, use the following query:

```
select ref_obj_id,ref_obj_ver from cdr_naming_versions where company_id =
your_company_idand obj_id = your_program_instance_obj_id and obj_ver=1;
```

Creating a Source Code Definition and Instance

You must create at least one Source Code object definition inside the Program definition to contain the PL/SQL packages you write.

Each Source Code definition must contain one and only one package. Each package can contain any number of functions and procedures, in any combination you choose.

It is not necessary to mark any Source Code as Primary because the Program is never executed as a whole.

See the following topic for additional information:

- Call the Create Source Code API

Call the Create Source Code API

To create a Source Code definition and an instance of it, call the API CDR_PUB_DF_SOURCEC)DE.CREATESOURCECODE.

Its signature is:

```
PROCEDURE CREATESOURCECODE(
     P_API_VERSION            IN     NUMBER,
     P_INIT_MSG_LIST          IN     VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_COMMIT                 IN     VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_VALIDATION_LEVEL       IN     NUMBER :=
CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
     X_RETURN_STATUS          OUT     VARCHAR2,
     X_MSG_COUNT              OUT     NUMBER,
     X_MSG_DATA               OUT     VARCHAR2,
     PIO_SCREF_SOURCECDRNAMING IN OUT    CDR_NAMING_VERSION_OBJ_TYPE,
     PI_CDRSCOBJTYPE           IN OUT    CDR_SRCCODE_OBJ_TYPE,
     PIO_CDRSCREFOBJTYPE       IN OUT    CDR_SRCCODE_REF_OBJ_TYPE,
     PI_CREATEOBJECT           IN     VARCHAR2,
     PI_DEFINITON_SUBTYPE_ID   IN     CDR_NAMINGS.OBJECT_SUBTYPE_ID%TYPE,
     PI_VLOBMODE               IN     VARCHAR2 := NULL,
     PIO_CDRSCBLOB             IN OUT    CDR_SRCCODE_BLOB_OBJ_TYPE,
     PIO_CDRSCCLOB             IN OUT    CDR_SRCCODE_CLOB_OBJ_TYPE
);
```

Enter Parameter values as follows:

- **PIO_SCREF_SOURCECDRNAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values that apply to the Source Code definition, as follows:

  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$SRCCDEREF'`
  - **name** = '`Enter_a_name_for_the_Source_Code`'
  - **namespace_obj_id** = `Enter_your_Program_Definition's_obj_id`
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - owning_location_rc = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null
  - **description** = `'Enter_a_Description_for_the_Source_Code`'
  - **copied_from_company_id** = null
  - **copied_from_obj_id** = null
  - **copied_from_obj_ver** = null
  - **ref_company_id** = null
  - **ref_obj_ver** = null
  - **object_version_number** = 1
  - **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  - **validation_status_rc** = null
  - **version_label** = null
- **PI_CDRSCOBJTYPE**. Enter CDR_SRCCODE_OBJ_TYPE values that apply to the Source Code definition, as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = 1
  - **tech_type_id** = `Enter_your_Tech_Type_ID`
  - **srccode_type_rc** = `'$FILETYPES$SQL'`
  - **shareable_flag_rc** = `'$YESNO$NO'`
  - **oracle_package_name** = `'Enter_the_package_name'`
  - **oracle_procedure_name** = `'Enter_the_name_of_the_procedure_inside_the_package`'
- **PIO_CDRSCREFOBJTYPE**. Enter CDR_SRCCODE_REF_OBJ_TYPE values that apply to the Source Code instance, as follows:
  - **company_id** = `Enter_your_company_ID`

  - **obj_id** = null
  - **obj_ver** = 1
  - **position** = 1
  - **primary_flag_rc** = `'$YESNO$NO'`

    Note:

    Only one Source Code in any Program can have its Primary flag set to Yes. In the case of an adapter Program, there is no true primary Source Code because the Program as a whole is never executed. The adapter calls the functions one at a time as needed. So you can set this flag to No for all Source Code definitions in the adapter.

  - **static_flag_rc** = `'$YESNO$NO'`
  - **static_program_company_id** = null
  - **static_program_obj_id** = null
    static_program_obj_ver_id = null
  - **fileref** = null
- **PI_CREATEOBJECT**. Enter "`BOTH`".
- **PI_DEFINITON_SUBTYPE_ID**. Null
- **PI_VLOBMODE**. Enter '`DIRECT`'.
- **PIO_CDRSCBLOB**. Null. You are uploading a PL/SQL package, which is a CLOB.
- **PIO_CDRSCCLOB**. This is a compound object of type CDR_SRCCODE_CLOB_OBJ_TYPE. Enter values as follows for one PL/SQL package containing the custom functions and procedures you have written for the adapter.
  - **file_name** = *`name_of_the_source_code_file`*
  - **file_clob** = *`source_code_text`*
  - **sc_obj_id** = null
  - **sc_obj_ver** = null

Creating a Variable

If you need user input to create a Load Set, Data Mart, Program, or Business Area—for example, an Oracle Remote Location name and connection—you must define a Parameter to collect the information. For each Parameter, you must define a Variable for the Parameter to reference.

For details, see the following:

- Call the Create Variable API
- Save the Variable ID for Future Use
- Check in the Variable

Call the Create Variable API

To create a Variable, call the API CDR_PUB_DF_VARIABLE.CREATEVARIABLE.

Its signature is:

```
PROCEDURE CREATEVARIABLE(
  P_API_VERSION              IN    NUMBER,
  P_INIT_MSG_LIST            IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT                   IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL         IN    NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS            OUT    VARCHAR2,
  X_MSG_COUNT                OUT    NUMBER,
  X_MSG_DATA                 OUT    VARCHAR2,
  PIO_NAMING                 IN OUT     CDR_NAMING_VERSION_OBJ_TYPE,
  PIO_VARIABLE               IN OUT     CDR_VAR_OBJ_TYPE,
  PI_DEFCLASSIFICATIONCOLL   IN    CDR_CLASSIFICATIONS_COLL
);
```

Enter Parameter values as follows:

- **PIO_NAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$CDRVAR'`
  - **name** = '`Enter_a_name_for_the_Variable`'

    Note:

    For the Name and Description attributes, enter a meaningful value appropriate for each Variable. This is important because you will probably create many Variables and you will need to reference them from Parameter definitions and, if you define Tables in your adapter, Table Columns.

  - **namespace_obj_id** = `Your_Adapter_Area's_obj_id`
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - **owning_location_rc** = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null
  - **description** = '`Enter_a_Description_for_the_Source_Code`'
  - **copied_from_company_id** = null

  - **copied_from_obj_id** = null
  - **copied_from_obj_ver** = null
  - **ref_company_id** = null
  - **ref_obj_ver** = null
  - **object_version_number** = 1
  - **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  - **validation_status_rc** = null
  - **version_label** = null
- **PIO_VARIABLE**. Enter CDR_VAR_OBJ_TYPE values that apply to the Source Code definition, as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = 1
  - **oracle_name** = *`'Enter_an_Oracle_Name'`*
  - **oracle_datatype_rc** = *`'Enter_one_of_the_valid_values'`*

    > **Note:**
    >
    > The valid Oracle_Datatype_RC values are:
    >
    > * `$ORADATATYPES$VARCHAR2`
    > * `$ORADATATYPES$NUMBER`
    > * `$ORADATATYPES$DATE`

  - **length** = *`Enter_the_Variable_length`*
  - **precision** = null (unless the variable is of data type number and requires a value for precision)
  - **sas_format** = If the data type is varchar2, enter `'$Char.length'`If the data type is number, enter `'$Num.length'`
    If you are creating an adapter to a SAS system, use the following default SAS formatting rules:Varchar2(10) becomes $10; Number(10,5) becomes 10.5;Date becomes datetime.
  - **sas_v6_name** = '*`Enter_SAS_v6_name`*'

    > **Note:**
    >
    > The SAS_V6_Name cannot be longer than 8 characters.

  - **sas_v8_name** = *`'Enter_SAS_v8_name'`*
  - **sas_label** = *`'Enter_Sas_Label'`*

  - **nullable_flag** = Enter `'$YESNO$YES'` or `'$YESNO$NO'` depending on whether or not you want the variable to be nullable or mandatory.
  - **default_value** = null

Save the Variable ID for Future Use

The input/output Parameter PIO_NAMING returns the Object ID (`obj_id`) and version number (obj_ver) of the Variable.

You will need this ID when you define a Parameter based on this Variable. Be careful to save a meaningful name with the ID, as you may have many Variables.

Note:

If you are creating Tables in your adapter, you can create Variables and Table Columns at the same time. You do not need to create Variables first.

Check in the Variable

You must check in the Variable so that other objects can reference it.

Call API CDR_PUB_DF_VARIABLE.CHECKIN. Enter values as follows:

- **PIO_BASEOBJECT**. Enter CDR_BASE_OBJ_TYPE values to identify the variable.
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = *`Enter_the_variable's_obj_id`*
  - **obj_ver** = 1
  - **object_version_number** = 1
  - **namespace_obj_id** = *`Enter_your_Adapter_Area's_obj_id`*
  - **namespace_obj_ver** = 1
- **PI_COMMENT** = null

Creating a Parameter

Use Parameters to allow users to enter values during the definition or execution of a Load Set, Data Mart, Program, or Business Area. You must create a defined Parameter object for each parameter required and handle the user input in your source code.

Parameter definitions are usually contained directly in the Adapter Area. Alternatively, if you are creating multiple adapters for a single external system and more than one of them use the same Parameter definition, you may want to create the Parameter definition directly in the Adapter Domain.

Note:

If you are creating an adapter that must upload files, define a Parameter with its Parameter Type set to either BINARY_FILE or TEXT_FILE, as appropriate, and create an instance of it in the runtime Parameter Set (see Creating a Parameter Set.) Create a second Parameter called TMP_BLOB_ID, for example, and create an instance of it in the same Parameter Set. Write code to upload the file to a temporary location and store the ID for the file as the valueof TMP_BLOB_ID. Use this value in your pre-execution function and execution function.

For details, see the following:

- Call the Create Parameter API
- Save the Parameter ID for Future Use
- Check in the Parameter

Call the Create Parameter API

To create a Parameter, call the API CDR_PUB_DF_PARAMETER.CREATEPARAMETER.

Its signature is:

```
PROCEDURE CREATEPARAMETER(
     P_API_VERSION             IN    NUMBER,
     P_INIT_MSG_LIST           IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_COMMIT                  IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
     P_VALIDATION_LEVEL        IN    NUMBER :=
CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
     X_RETURN_STATUS           OUT    VARCHAR2,
     X_MSG_COUNT               OUT    NUMBER,
     X_MSG_DATA                OUT    VARCHAR2,
     PIO_PARAMNAMING           IN OUT    CDR_NAMING_VERSION_OBJ_TYPE,
     PIO_CDRPARAMOBJTYPE       IN OUT    CDR_PARAMETER_OBJ_TYPE,
     PI_CREATE_OBJECT          IN    VARCHAR2,
     PI_INSTANCE_SUBTYPE_ID    IN    CDR_NAMINGS.OBJECT_SUBTYPE_ID%TYPE,
     PI_PARENTNAMING           IN OUT    CDR_BASE_OBJ_TYPE,
     PO_DEFCLASSIFICATIONCOLL  IN    CDR_CLASSIFICATIONS_COLL
);
```

Enter parameter values as follows:

- **PIO_PARAMNAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$PARAMETER'`
  - **name** = '`Enter_a_name_for_the_Parameter`'

> **Note:**
>
> For the Name and Description attributes, enter a meaningful value appropriate for each Parameter. This is important because you will probably create many Parameters and you will need to create instances of them.

- **namespace_obj_id** = `Enter_your_Adapter_Area's_obj_id`
- **namespace_obj_ver** = 1
- **namespace_start_obj_ver** = 1
- **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
- **owning_location_rc** = null
- **checked_out_flag_rc** = `'$YESNO$NO'`
- **checked_out_id** = null
- **object_subtype_id** = null
- **description** = '*`Enter_a_Description_for_the_Parameter`*'
- **copied_from_company_id** = null
- **copied_from_obj_id** = null
- **copied_from_obj_ver** = null
- **ref_company_id** = *`Enter_your_company_ID`*
- **ref_obj_id** = *`Enter_the_obj_ID_of_the_Variable_this_Parameter_references`*
- **ref_obj_ver** = 1
- **object_version_number** = 1
- **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
- **validation_status_rc** = null
- **version_label** = null

• **PIO_CDRPARAMOBJTYPE**. Enter CDR_PARAMETER_OBJ_TYPE values that apply to the Parameter definition, as follows:

- **company_id** = *`Enter_your_company_ID`*
- **obj_id** = null
- **obj_ver** = 1
- **prompt** = *`'Enter_the_label_you_want_to_appear_in_the_UI'`*
- **allowed_values_rc** (Required) This attribute determines what type of value the Parameter will support. For further information, see the chapter on Parameters in the *Oracle LSH Application Developer's Guide*. The allowed attribute values are:
  * $PARAMALLOWVALS$PGMGENLOV (Program-generated list of values)
  * $PARAMALLOWVALS$STATICLOV (Static list of values)

    * $PARAMALLOWVALS$SINGLEVALUE (Single value)

– **lov_company_id** = null
– **lov_id** = null
– **lov_ver** = null
– **lov_prg_inst_company_id** = null or, if you are defining a programmatically generated list of values, enter the company_id of the Program instance that you need to run to generate the LOV.
– **lov_prg_inst_id** = null or, if you are defining a programmatically generated list of values, enter the obj_id of the Program instance that you need to run to generate the LOV.
– **lov_prg_inst_ver** = null or, if you are defining a programmatically generated list of values, enter the obj_ver of the Program instance that you need to run to generate the LOV.
– **lov_sc_ref_company_id** = null or, if you are defining a programmatically generated list of values, enter the company_id of the relevant Source Code instance in the Program instance.
– **lov_sc_ref_id** = null or, if you are defining a programmatically generated list of values, enter the obj_id of the relevant Source Code instance in the Program instance.
– **lov_sc_ref_ver** = null or, if you are defining a programmatically generated list of values, enter the obj_ver of the relevant Source Code instance in the Program instance.
– **lov_cla_level_id** = null or, if you are defining a list of values based on terms in a classification hierarchy level, enter the level_id of the of the relevant level.
– **lov_default_cla_id** = null
– **lov_multi_flag_rc** Enter `$YESNO$YES` if the Parameter supports either a static list of values or a program-generated list of values and you want to support selecting more than one value at a time in the user interface.
  Enter `$YESNO$NO` if the Parameter supports only a single value.
– **validation_rule_rc** Enter one of the following values:
    * `$VALDNRULES$NONE` (no validation rule defined)
    * `$VALDNRULES$USEALLOWEDVALS` (the parameter's value will be validated against the list of values defined for the parameter, either Programatic or Static)
    * `$VALDNRULES$PROGRAMMATIC` (the parameter's value will be validated against a list of values generated by source code different from the source code that generates the Parameter's list of values—if any)
– **val_prg_inst_company_id** = null or, if you are validating user-entered values programmatically, enter the company_id of the Program instance that you run to perform the validation.
– **val_prg_inst_id** = null or, if you are validating user-entered values programmatically, enter the obj_id of the Program instance that you run to perform the validation.

  - **val_prg_inst_ver** = null or, if you are validating user-entered values programmatically, enter the obj_ver of the Program instance that you run to perform the validation.
  - **val_sc_ref_company_id** = null or, if you are validating user-entered values programmatically, enter the company_id of the relevant Source Code instance in the Program instance.
  - **val_sc_ref_id** = null or, if you are validating user-entered values programmatically, enter the obj_id of the relevant Source Code instance in the Program instance.
  - **val_sc_ref_ver** = null or, if you are validating user-entered values programmatically, enter the obj_ver of the relevant Source Code instance in the Program instance.
  - **input_output_rc** = `'$PARAMDIRECTS$INOUT'`
  - **read_only_flag_rc** = `'$YESNO$NO'`
  - **visible_flag_rc** = `'$YESNO$YES'`
  - **mandatory_flag_rc** = `'$YESNO$YES'`
  - **default_value** = null (or enter a default value if you want one)
  - **position** = null
  - **param_type_rc** = null

    Note:

    By default the system sets the Parameter type to Scalar.

  - **auto_share_field_flag_rc** = null
- **PI_CREATEOBJECT** Enter `DEFN`.
- **PI_INSTANCE_SUBTYPE_ID** = null
- **PI_PARENTNAMING** Enter CDR_BASE_OBJ_TYPE values that apply to your Adapter Area as follows:
  - **company_id** = *`Enter_the_company_id_of_your_Adapter_Area`*
  - **obj_id** = *`Enter_the_obj_id_of_your_Adapter_Area`*
  - **obj_ver** = 1
- **PO_DEFCLASSIFICATIONCOLL** Null

Save the Parameter ID for Future Use

The input/output Parameter PIO_PARAMNAMING returns the Object ID (`obj_id`) and version number (obj_ver) of the Parameter.

You will need this ID when you define a Parameter instance based on this Parameter. Be careful to save a meaningful name with the ID, as you may have many Parameters.

Check in the Parameter

You must check in the Parameter so that other objects can reference it.

Call API CDR_PUB_DF_PARAMETER.CHECKINPARAMETER. Its signature is:

```
PROCEDURE CHECKINPARAMETER(
  P_API_VERSION       IN    NUMBER,
  P_INIT_MSG_LIST     IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT            IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL  IN    NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS     OUT    VARCHAR2,
  X_MSG_COUNT         OUT    NUMBER,
  X_MSG_DATA          OUT    VARCHAR2,
  PIO_CDRNAMING       IN OUT    CDR_BASE_OBJ_TYPE,
  PI_COMMENT          IN    VARCHAR2
);
```

Enter values as follows:

- **PIO_BASEOBJECT**. Enter CDR_BASE_OBJ_TYPE values to identify the variable.
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = *`Enter_the_Parameter's_obj_id`*
  - **obj_ver** = 1
  - **object_version_number** = 1
  - **namespace_obj_id** = *`Enter_your_Adapter_Area's_obj_id`*
  - **namespace_obj_ver** = 1
- **PI_COMMENT** = null

Creating a Parameter Set

You may create one, two, or three Parameter Sets to collect user input during the definition, installation, or execution of a Load Set, Data Mart, or Program, or the launch of a visualization tool.

You must give these Parameter Sets specific names and attribute values so that the system can use them properly; see Planning Parameters and Parameter Sets.

For more details, see the following:

- Call the Create Parameter Set API
- Save the Parameter Set ID for Future Use

Call the Create Parameter Set API

To create a Parameter, call the API CDR_PUB_DF_PARAMETER_SET.CREATEPARAMETERSET.

Its signature is:

```
PROCEDURE CREATEPARAMETERSET(
  P_API_VERSION         IN    NUMBER,
  P_INIT_MSG_LIST       IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT              IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL    IN    NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS       OUT   VARCHAR2,
  X_MSG_COUNT           OUT   NUMBER,
  X_MSG_DATA            OUT   VARCHAR2,
  PIO_SOURCECDRNAMING   IN OUT    CDR_NAMING_VERSION_OBJ_TYPE,
  PI_CDRPSOBJTYPE       IN    CDR_PARAM_SETS_OBJ_TYPE,
  PI_CREATEOBJECT       IN    VARCHAR2,
  PI_INSTANCE_SUBTYPE_ID    IN   CDR_NAMINGS.OBJECT_SUBTYPE_ID%TYPE,
  PI_DEFCLASSIFICATIONCOLL  IN   CDR_CLASSIFICATIONS_COLL
);
```

Enter parameter values as follows:

- **PIO_PARAMNAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = null
  - **name** = '`Enter_one_of_the_required_names_for_the_Parameter_Set`'

    Note:

    The valid values are:

    * `PARAMETERSET_LOADSETLEVEL_DEF`
    * `PARAMETERSET_OPERATORLEVEL`
    * `PARAMETERSET_LOADSETLEVEL_RUN`

  - **namespace_obj_id** = `Enter_your_Adapter_Area's_obj_id`
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - **owning_location_rc** = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null
  - **description** = '`Enter_the_same_value_that_you_entered_for_the_Parameter_Set_name`'
  - **copied_from_company_id** = null
  - **copied_from_obj_id** = null

  - **copied_from_obj_ver** = null
  - **ref_company_id** = `Enter_your_company_ID`
  - **ref_obj_id** = `Enter_the_obj_ID_of_the_Variable_this_Parameter_references`
  - **ref_obj_ver** = 1
  - **object_version_number** = 1
  - **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  - **validation_status_rc** = null
  - **version_label** = null
- **PI_CDRPSOBJTYPE**. Enter CDR_PARAM_SETS_OBJ_TYPE values that apply to the Parameter Set definition, as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = 1
  - **usage** = `'Enter_the_usage_value_required_for_Parameter_Sets_with_this_name'`

    Note:

    The valid values are:

    * `DEFINITION` for Parameter Sets named `PARAMETERSET_LOADSETLEVEL_DEF`
    * `OPERATOR` for Parameter Sets named `PARAMETERSET_OPERATORLEVEL`
    * `EXECUTION` for Parameter Sets named `PARAMETERSET_LOADSETLEVEL_RUN`

  - **pr_ref_id** = null
  - **pr_ref_ver** = null
  - **parameter_set_type_rc** = null

Save the Parameter Set ID for Future Use

You will need this ID for the namespace_obj_id when you create Parameter instances inside this Parameter Set.

Oracle Life Sciences Data Hub creates standard IDs for Adapter Parameter Sets:

- `PS1ID` for Parameter Sets named `PARAMETERSET_LOADSETLEVEL_DEF`
- `PS2ID` for Parameter Sets named `PARAMETERSET_OPERATORLEVEL`
- `PS3ID` for Parameter Sets named `PARAMETERSET_LOADSETLEVEL_RUN`

Creating a Parameter Instance in a Parameter Set

For each Parameter definition you have created, create a Parameter instance in the appropriate Parameter Set.

For more details, see the following:

- Call the Create Parameter API

Call the Create Parameter API

To create a Parameter, call the API CDR_PUB_DF_PARAMETER.CREATEPARAMETER.

Enter parameter values as follows:

```
PROCEDURE CREATEPARAMETER(
  P_API_VERSION        IN    NUMBER,
  P_INIT_MSG_LIST      IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT             IN    VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL   IN    NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS      OUT   VARCHAR2,
  X_MSG_COUNT          OUT   NUMBER,
  X_MSG_DATA           OUT   VARCHAR2,
  PIO_PARAMNAMING      IN OUT         CDR_NAMING_VERSION_OBJ_TYPE,
  PIO_CDRPARAMOBJTYPE  IN OUT         CDR_PARAMETER_OBJ_TYPE,
  PI_CREATE_OBJECT     IN    VARCHAR2,
  PI_INSTANCE_SUBTYPE_ID    IN        CDR_NAMINGS.OBJECT_SUBTYPE_ID%TYPE,
  PI_PARENTNAMING           IN OUT    CDR_BASE_OBJ_TYPE,
  PO_DEFCLASSIFICATIONCOLL  IN        CDR_CLASSIFICATIONS_COLL
);
```

- **PIO_PARAMNAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values as follows:
  - **company_id** = `Enter_your_company_ID`
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$PARAMREF'`
  - **name** = '`Enter_a_name_for_the_Parameter_instance`'
  - **namespace_obj_id** = `Enter_the_Parameter_Set_ID`
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - **owning_location_rc** = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null

  - **description** = '*`Enter_a_Description_for_the_Parameter_instance`*'
  - **copied_from_company_id** = null
  - **copied_from_obj_id** = null
  - **copied_from_obj_ver** = null
  - **ref_company_id** = *`Enter_your_company_ID`*
  - **ref_obj_id** = *`Enter_the_obj_ID_of_the_Parameter_definition_this_Parameter_instance_references`*
  - **ref_obj_ver** = 1
  - **object_version_number** = 1
  - **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  - **validation_status_rc** = null
  - **version_label** = null
- **PIO_CDRPARAMOBJTYPE**. Do not enter any values.
- **PI_CREATEOBJECT**. Enter `INST`.
- **PI_INSTANCE_SUBTYPE_ID**. Null
- **PI_PARENTNAMING**. Enter the following CDR_BASE_OBJ_TYPE values. The first four apply to the Parameter Set into which you are putting the Parameter instance. The last two (namespace) attributes apply to the Adapter Area, which is the parent of the Parameter Set.
  - **company_id** = Enter your company ID; see Getting Your Company ID.
  - **obj_id** = Enter the Object ID of the Parameter Set.
  - **obj_ver** = 1
  - **object_version_number** = 1
  - **namespace_obj_id** = Enter the Object ID of the Adapter Area.
  - **namespace_obj_ver** = 1
- **PO_DEFCLASSIFICATIONCOLL**. Null

Creating a Table Definition

If you are creating a Load Set adapter and the external source data system has fixed data structures, you may want to define those data structures as Tables in the Adapter Area rather than uploading them or forcing the user to create them manually each time a user creates a Load Set.

This enhances performance, reduces the possibility of error, and eliminates the need to connect to a remote database during Load Set definition.

You can either create a Table Descriptor for every Table definition in every Load Set of this type, or you can create a list of values and allow the person defining the Load Set to select which Table Descriptors he or she wants.

To create a list of values:

- Write a procedure to retrieve a list of all the Table definitions in the Adapter Area and to insert them into a list values so that the user can select some or all of them to be loaded by the Load Set.
- When you create the Adapter Area, enter the procedure's name as the value for auto_add_tab_desc_function.

> **Note:**
>
> The above description is correct. The intended functions for the above column and the Auto_Add_Tab_Desc_LOV column are reversed.

- When you create the Adapter Area, set allow_auto_add_tab_desc to YES.

In addition, you must:

- Write a procedure to create Table Descriptors in Oracle Life Sciences Data Hub based on each Table definition required.
- When you create the Adapter Area, enter this procedure's name as the value for auto_add_tab_desc_lov.

> **Note:**
>
> Data Mart and Business Area adapters do not require Table definitions because their source tables are within Oracle Life Sciences Hub.

For more details, see the following:

- Call the Create Table API
- Call the Create Column API

Call the Create Table API

To create a Table definition, call the API CDR_PUB_DF_TABLE.CREATETABLEDEFINITION.

Enter parameter values as follows:

```
PROCEDURE CREATETABLEDEFINITION(
  P_API_VERSION        IN     NUMBER,
  P_INIT_MSG_LIST      IN     VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT             IN     VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL   IN     NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS      OUT    VARCHAR2,
  X_MSG_COUNT          OUT    NUMBER,
  X_MSG_DATA           OUT    VARCHAR2,
  PIO_NAMING           IN OUT      CDR_NAMING_VERSION_OBJ_TYPE,
  PIO_TABLE            IN OUT      CDR_TABLE_OBJ_TYPE,
  PI_INSTANCESUBTYPEID        IN     NUMBER,
  PI_DEFCLASSIFICATIONCOLL    IN     CDR_CLASSIFICATIONS_COLL,
  PI_INSTCLASSIFICATIONCOLL   IN     CDR_CLASSIFICATIONS_COLL
);
```

- **PIO_NAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values that apply to the Program definition, as follows:
  – **company_id** = `Enter_your_company_ID`
  – **obj_id** = null
  – **obj_ver** = null
  – **object_type_rc** = `'$OBJTYPES$TABLE'`
  – **name** = '`Enter_a_name_for_the_Table`'
  – **namespace_obj_id** = `Enter_your_Adapter_Area's_obj_id`
  – **namespace_obj_ver** = 1
  – **namespace_start_obj_ver** = 1
  – **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  – **owning_location_rc** = null
  – **checked_out_flag_rc** = `'$YESNO$NO'`
  – **checked_out_id** = null
  – **object_subtype_id** = null
  – **description** = '`Enter_a_Description_for_the_Table`'
  – **copied_from_company_id** = null
  – **copied_from_obj_id** = null
  – **copied_from_obj_ver** = null
  – **ref_company_id** = null
  – **ref_obj_ver** = null
  – **object_version_number** = 1
  – **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  – **validation_status_rc** = null
  – **version_label** = null
- **PI_TABLE**. Enter CDR_TABLE_OBJ_TYPE values as follows:
  – **company_id** = `Enter_your_company_ID`
  – **obj_id** = null
  – **obj_ver** = 1
  – **oracle_name** = `'Enter_an_Oracle_name_for_the_Table'`
  – **sas_name** = `'Enter_a_SAS_name_for_the_Table'`
  – **sas_label** = `'Enter_a_SAS_label_for_the_Table'`
  – **sas_v6_flag** = `'$YESNO$YES'` if you are using SAS v6 or `'$YESNO$NO'` if you are using a more recent SAS version
  – **audit_tabc_company_id** = null
  – **audit_tabc_obj_id** = null
  – **audit_tabc_obj_ver** = null

  - **snapshot_flag_rc** = *`'set_to_$YESNO$YES_to_allow_snapshots or $YESNO$NO_prevent_them'`*
  - **process_type_rc** = Valid values are: $PROCESSTYPES$RELOAD (Reload), $PROCESSTYPES$STAGINGWAUDIT (Staging with Audit), $PROCESSTYPES$STAGINGWOAUDIT (Staging without Audit), $PROCESSTYPES$TRANSWOAUDIT (Transactional without Audit), $PROCESSTYPES$TRANSWAUDIT (Transactional with Audit)
  - **blinding_flag_rc** = *`Enter_$YESNO$YES_if_the_adapter_will_load_blinded_data_or_$YESNO$NO_if_it_will_not`*
- **PI_INSTANCESUBTYPEID**. Null
- **PI_DEFCLASSIFICATIONCOLL**. Null
- **PI_INSTCLASSIFICATIONCOLL**. Null

Call the Create Column API

Call the API CDR_PUB_DF_TABLE.CREATECOLUMN to create Columns for the Table, one at a time.

You can create both the Column and the Variable on which it is based at the same time.

- **PIO_NAMING**. Enter CDR_NAMING_VERSION_OBJ_TYPE values as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = null
  - **object_type_rc** = `'$OBJTYPES$COLUMN'`
  - **name** = '*`Enter_a_name_for_the_Column`*'
  - **namespace_obj_id** = *`Enter_the_Table's_obj_ID`*
  - **namespace_obj_ver** = 1
  - **namespace_start_obj_ver** = 1
  - **namespace_end_obj_ver**= `cdr_def_constants.cdr_max_def_object_version`
  - **owning_location_rc** = null
  - **checked_out_flag_rc** = `'$YESNO$NO'`
  - **checked_out_id** = null
  - **object_subtype_id** = null
  - **description** = '*`Enter_a_Description_for_the_Column`*'
  - **copied_from_company_id** = null
  - **copied_from_obj_id** = null
  - **copied_from_obj_ver** = null
  - **ref_company_id** = null
  - **ref_obj_ver** = null

  - **object_version_number** = 1
  - **status_rc** = `'$NAMING_STATUS$INSTALLABLE'`
  - **validation_status_rc** = null
  - version_label = null
- **PIO_VARIABLE**. Enter CDR_VAR_OBJ_TYPE values to define the Variable as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = 1
  - **oracle_name** = *`'Enter_an_Oracle_name_for_the_Table'`*
  - **oracle_datatype** = *`'Enter one: $ORADATATYPES$VARCHAR2, $ORADATATYPES$NUMBER, $ORADATATYPES$DATE'`*
  - **length** = *`'Enter_a_length_for_the_Table'`*
  - **precision** = *`'If_your_Variable's_datatype_is_NUMBER,_enter_its_precision'`*
  - **sas_v6_name** = *`'Enter_your_variable's_SASv6_name_(up_to_8_chars)'`*
  - **sas_v8_name** = *`'Enter_your_variable's_SASv8_name_(up_to_32_chars)'`*
  - **sas_label** = *`'Enter_your_variable's_SAS_label_(up_to_256_chars)'`*
  - **sas_format** = *`'Enter_your_variable's_SAS_format'`*
  - **nullable_flag** = *`'Enter_$YESNO$YES_if_the_value_can_be_null_or_$YESNO$NO_if_not'`*
  - **default value** = *`Set_to_Null_or_enter_a_default_value_for_the_variable`*
- **PIO_COLUMN**. Enter CDR_COLUMNS_OBJ_TYPE values to define the Variable as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = 1
  - **position_number** = *`'Enter_your_column's_position_number_in_the_Table'`*
  - **nullable_flag** = *`'Enter_$YESNO$YES_if_the_value_can_be_null_or_$YESNO$NO_if_not'`*
- **PI_CREATETYPE** Enter `BOTH` to create a Variable and a Column at the same time.
- **PI_DEFCLASSIFICATIONCOLL** Null

Installing the Work Area

After you have created the Adapter Domain and all the definitional objects within it, install the Work Area by calling the API

CDR_PUB_DF_WORKAREA.INSTALLWACONTROLLER and the Program instance it contains.

This API also checks in the Program instance and its definition.

Note:

If the Program is checked out, which it normally is at this point, and the person running this API is different from the person who created the Program, Work Area installation fails. The person who created(or most recently checked out) the Program definition must either explicitly check in the Program or run the Work Area installation. See Checking In Objects and Setting Their Validation Status for further information.

To create a Work Area, call the API CDR_PUB_DF_WORKAREA.CREATEWORKAREA. Its signature is:

```
PROCEDURE INSTALLWACONTROLLER(
  P_API_VERSION       IN     NUMBER,
  P_INIT_MSG_LIST     IN     VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_COMMIT            IN     VARCHAR2 := CDR_PUB_DEF_CONSTANTS.G_FALSE,
  P_VALIDATION_LEVEL  IN     NUMBER := CDR_PUB_DEF_CONSTANTS.G_VALID_LEVEL_FULL,
  X_RETURN_STATUS     OUT    VARCHAR2,
  X_MSG_COUNT         OUT    NUMBER,
  X_MSG_DATA          OUT    VARCHAR2,
  PIO_OWABASENAMING   IN OUT    CDR_BASE_OBJ_TYPE,
  PI_VINSTALLMODE     IN    CDR_INSTALLATIONS.INSTALLATION_MODE_RC%TYPE,
  PI_VFORCEREGEN      IN    CDR_INSTALLATIONS.FORCE_REGEN_FLAG_RC%TYPE,
  PI_VBATCH           IN    CDR_INSTALLATIONS.BATCH_FLAG_RC%TYPE,
  PI_VACTION          IN    CDR_INST_ELEMENTS.INSTALL_ACTION_RC%TYPE,
  PI_COINSTDETAILS    IN    CDR_INSTALLATION_DETAILS_COLL
);
```

Enter values as follows:

- **PIO_OWABASENAMING**. Enter CDR_BASE_OBJ_TYPE values as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = *`Enter_the_Work_Area's_obj_ID`*
  - **obj_ver** = *`Enter_the_Work_Area's_obj_ver`*
  - **object_version_number** = *`Get_this_value_from_cdr_df_naming_v`*
  - **namespace_obj_id** = *`Enter_your_Adapter_Area's_object_ID`*
  - **namespace_obj_ver** = *`Enter_your_Adapter_Area's_obj_ver`*
- **PI_VINSTALLMODE**. Enter `'$INSTALLMODE$UPGRADE'`
- **PI_VFORCEREGEN**. Enter `'$YESNO$YES'`
- **PI_VBATCH**. Enter `'$YESNO$NO'`
- **PI_VACTION**. Enter one of the following values:
  - `'$INSTALLCMD$COMPLETE'` Enter this value if this is the first time the Work Area is being installed, or if the last installation was successful, or if the last installation failed and you want to continue from the last successfully completed phase.

  - `'$INSTALLCMD$CANCEL'` Enter this value if the last installation failed and you want to begin the installation process from the beginning.

    > **Note:**
    >
    > You must then run the API again with PI_VACTION set to `'$INSTALLCMD$COMPLETE'`.

- **PI_COINSTDETAILS**. This is a collection of CDR_INST_DET_OBJ_TYPEs. For each object in the Work Area that you want to install, initialize a CDR_INST_DET_OBJ_TYPE and then extend the collection. Normally an adapter has only a single Program instance in the Work Area. Enter values to identify it as follows:
  - **company_id** = *`Enter_your_company_ID`*
  - **obj_id** = null
  - **obj_ver** = 1
  - **object_type_rc** = For Programs, enter `'$OBJTYPES$PROGRAMINST'`.
  - **omit_from_install_flag_rc** = `'$YESNO$NO'`
  - **install_action_rc** = `'$INSTOBJACT$REPLACE'`

5

Using the Generic Visualization Adapter

Oracle Life Sciences Data Hub includes a generic visualization adapter that allows you to integrate an external tool with Oracle Life Sciences Hub to view data without building a custom adapter.

To use this adapter, you do not need to follow instructions in any section of this guide other than this chapter.

To integrate your visualization tool with Oracle Life Sciences Hub:

- Use an Oracle Life Sciences Hub view to display, when a user logs in to the visualization tool, the Business Area instances to which the user has access; see Display User's Business Area Instances.
- Use Oracle Life Sciences Hub APIs to allow the user to select appropriate blinding and currency settings; see Getting Possible Blinding Types of a Business Area Instance and Getting Snapshot Labels Common to All Tables in a BA Instance for a Given Blinding Access Type.
- Use Oracle LSH APIs to initialize the user's selected Business Area instance with those settings and to reset as required; see Initializing the Business Area Instance and Resetting a Generic Visualization Business Area.
- Use an Oracle Life Sciences Hub view to Retrieve Table Instance Details for the data you want to display.

To use the adapter, Definers create Business Areas of type Generic Visualization. When a Definer installs a Generic Visualization Business Area instance for the first time, the adapter creates a database schema exclusively for the Business Area instance. As with other Business Areas, data in Table instances mapped to the Business Area's Table Descriptors can be viewed in the visualization tool. Normally users log in directly to the visualization tool through its URL.

Note:

You do not need to define any service locations or service instances for this adapter.

For more information, see the following:

- Generic Visualization Adapter APIs
  The adapter includes public APIs that you can call from the external visualization system.
- Generic Visualization Adapter Views
  You can retrieve data for a user's business area instances and for table instances.
- Generic Visualization Business Area Instance Attributes
  Business Area instances of type Generic Visualization have several attributes including Schema Name, Default Currency, and Default Blinding Access Type.

- Generic Visualization Adapter Security
  Security for the generic visualization adapter has several components including Database and User Accounts, Object Security, Data Blinding and Currency, and Auditing.
- Generic Visualization Adapter Definitional Components
  All the defined objects, from the Adapter Domain down, are shipped with Oracle Life Sciences Data Hub.

Generic Visualization Adapter APIs

The adapter includes public APIs that you can call from the external visualization system.

For more information about using APIs, see the *Oracle Life Sciences Data Hub Application Programming Interface Guide*.

For additional details, see the following:

- Initializing the Business Area Instance
- Resetting a Generic Visualization Business Area
- Getting Possible Blinding Types of a Business Area Instance
- Getting Snapshot Labels Common to All Tables in a BA Instance for a Given Blinding Access Type

Initializing the Business Area Instance

Use the API CDR_PUB_API_GVA.SETINITIALIZEBA to initialize a particular Generic Visualization Business Area instance with a given currency and blinding access type.

This API also looks for an Oracle Life Sciences Data Hub user account linked to the database account with which the user logged in. If there is a linked user account, the API enforces the security privileges of the user account; if not, the API enforces the security privileges of the database account.

If the currency and the blinding access values are not set by the user, the API uses the default values set in the Business Area instance properties and the user's privileges to determine the data to display; see Generic Visualization Business Area Instance Attributes and Generic Visualization Adapter Security.

See the following:

- Initializing a Business Area Instance Repeatedly in the Same Session
- Initializing Multiple Business Area Instances in the Same Session

Initializing a Business Area Instance Repeatedly in the Same Session

You can invoke this API multiple times on the same Business Area instance to change the currency and blinding access types in a single user session.

Initializing Multiple Business Area Instances in the Same Session

You can also invoke this API multiple times to allow a user to read data from multiple Business Area instances as long as the user views either real or dummy data across all Business Areas. The Real (BlindBreak) and Real (Unblinded) blinding access types are considered as reading real data while NA/Dummy is considered as reading dummy data.

If the user selects a blinding access type for a Business Area that is incompatible with the blinding access types selected for other Business Areas in the same session, the API errors out with the message, "There is a change in reading dummy data to blinded data or vice-versa. Please reset access to all Business Areas using resetBAAccess api and try again;" see Resetting a Generic Visualization Business Area.

The following settings and combinations of settings work:

- **NA/Dummy** blinding access type on all Business Areas: The user sees only dummy data in blinded Table instances.
- **Real (Unblinded)** blinding access type on all Business Areas: The user sees unblinded data in all Business Areas. This option is available only if all the Business Area's Table instances whose Blinding flag is set to Yes have a Blinding Status of Unblinded.
- **Real (Blind Break)** blinding access type on all Business Areas: The user sees currently blinded data in blinded Table instances in all Business Areas.
- **Real (Unblinded)** and **Real (Blind Break)** blinding access types: The user sees unblinded data in Business Areas where this option is available and currently blinded data in others.

> **Note:**
>
> The user must always have the appropriate blinding-related privileges. Without them the user can see only dummy data in Table instances whose Blinding flag is set to **Yes**. Table instances whose blinding flag is set to **No** contain data that was never blinded (NA for Not Applicable) and is always available.

Signature

```
PROCEDURE SETINITILIZEBA(
                                                        PI_COMPANYID IN
CDR_NAMINGS.COMPANY_ID%TYPE,
                                                        PI_OBJID     IN
CDR_NAMINGS.OBJ_ID%TYPE,
                                                        PI_OBJVER    IN
CDR_NAMING_VERSIONS.OBJ_VER%TYPE,
                                                        PI_VCURRENCY IN VARCHAR2,
                                                        PI_VBLINDINGACCESSTYPE    IN
VARCHAR2,
                                                        X_RETURN_STATUS     OUT
NOCOPY  VARCHAR2,
                                                        X_MSG_COUNT         OUT
NOCOPY  NUMBER,
                                                        X_MSG_DATA          OUT
NOCOPY  VARCHAR2
```

Parameters

This API has the following parameters:

PI_COMPANYID. Enter the Business Area instance's company ID.

PI_OBJID. Enter the Business Area instance's object ID.

PI_OBJVER. Enter the Business Area instance's object version.

PI_VCURRENCY. Enter the currency value. The allowed values are `Current` or any snapshot label common to all Table instances mapped to the Business Area's Table Descriptors.

PI_VBLINDINGACCESSTYPE. Enter the blinding access type. The allowed values are: `NA/Dummy` , `Real(Unblinded)`, or `Real(BlindBreak)`. Note that there is no space between `Real` and the parentheses/brackets.

Resetting a Generic Visualization Business Area

Use public API CDR_PUB_API_GVA.RESETBAACCESS to clear all the initializations of Business Area schemas.

It is equivalent to logging out and logging back in to the system.

This API has no parameters.

Users cannot select incompatible blinding access types for different Business Areas in the same session. If they do, the initialization API errors out with a message to call this API.

You may want to trap the Initialization API's error message and, if possible, display a dialog box warning the user that the blinding access type is incompatible with open Business Areas (or the visualization tool's equivalent) and give the user the following options:

- Continue with the current setting, which will result in closing all other Business Areas. If the user selects this option, run the Reset API and then the Initialization API with the requested setting for the current Business Area.
- Change the setting for the current Business Area to be compatible with those already open. If the user selects this option, run the Initialization API with the appropriate setting.

Alternatively, display the Initialization API's error message and provide a way in the UI to invoke the Reset API to close all open Business Areas. Or trap the Initialization API's error message, invoke the Reset API to close all open Business Areas, and display a message. In either case, the user can then select a Business Area and blinding access type as required.

Getting Possible Blinding Types of a Business Area Instance

You can allow users with the necessary privileges to specify if they want to view only nonblinded and dummy data, or data that includes unblinded or currently blinded data; see Generic Visualization Adapter Security.

Use public API CDR_PUB_API_GVA.GETBAVALIDBLINDINGACCESSTYPES to get the possible blinding access types of a Business Area instance in the current session, based on the blinding statuses of underlying Business Area Table instances and the user's privileges.

Note:

The option to read unblinded data is available only if **all** relevant Table instances—that is, all Table instances that are mapped to the Business Area instance's Table Descriptors and have their Blinding Flag set to **Yes**—have a Blinding Status of **Unblinded**.

If you do not use the API or the user does not provide a selection, the system displays the data specified for the Business Area instance; see Default Blinding Access Type.

Signature

```
FUNCTION GETBAVALIDBLINDINGACCESSTYPES(

PI_COMPANYID IN CDR_NAMINGS.COMPANY_ID%TYPE,

PI_OBJID     IN CDR_NAMINGS.OBJ_ID%TYPE,

PI_OBJVER    IN CDR_NAMING_VERSIONS.OBJ_VER%TYPE)
RETURN BLINDINGACCESSTYPESCOLL PIPELINED;
```

Return

A collection (BLINDINGACCESSTYPESCOLL) of the possible blinding access types. The possible values are: `NA/Dummy` , `Real(Unblinded)`, or `Real(BlindBreak)`. Note that there is no space between `Real` and the parentheses/brackets.

Parameters

This API has the following parameters:

PI_COMPANYID. Enter the Business Area instance's company ID.

PI_OBJID. Enter the Business Area instance's object ID.

PI_OBJVER. Enter the Business Area instance's object version.

BLINDINGACCESSTYPESCOLL. This is the list of possible blinding access types.

Getting Snapshot Labels Common to All Tables in a BA Instance for a Given Blinding Access Type

You can allow the user, on login, to choose to view current data or data based on a snapshot label.

A snapshot label is available for selection only if all source Table instances have the same label applied. Table instances that are pass-through views can display only current data, so if one or more Table instance is a view, only current data will be available for the Business Area.

Use public API CDR_PUB_API_GVA.GETSNAPSHOTLABELS to get the snapshot labels common to all Table instances mapped to Table Descriptors in a single Business Area instance. The API has an input parameter for blinding access type because snapshot labels may be different for dummy and real data.

If you do not use the API or the user does not provide a snapshot label selection, the system displays the data currency specified for the Business Area; see Default Currency.

Signature

```
FUNCTION GETSNAPSHOTLABELS(
                                                            PI_COMPANYID IN
CDR_NAMINGS.COMPANY_ID%TYPE,
                                                            PI_OBJID     IN
CDR_NAMINGS.OBJ_ID%TYPE,
                                                            PI_OBJVER    IN
CDR_NAMING_VERSIONS.OBJ_VER%TYPE,
                                                            PI_VBLINDINGACCESSTYPE
IN VARCHAR2)
RETURN CURRENCYCOLL PIPELINED;
```

Return

A collection (CURRENCYCOLL) of the snapshot labels for a particular blinding access type in the Business Area instance common to all Tables within a Business Area Instance.

Parameters

This API has the following parameters.

PI_COMPANYID. Enter the Business Area instance's company ID.

PI_OBJID. Enter the Business Area instance's object ID.

PI_OBJVER. Enter the Business Area instance's object version.

PI_VBLINDINGACCESSTYPE. Enter the blinding access type. The allowed values are: `NA/Dummy` , `Real(Unblinded)`, or `Real(BlindBreak)`. Note that there is no space between `Real` and the parentheses/brackets.

Generic Visualization Adapter Views

You can retrieve data for a user's business area instances and for table instances.

For additional details, see the following:

- Display User's Business Area Instances
- Retrieve Table Instance Details

Display User's Business Area Instances

Use naming view CDR_PUB_GENERIC_BA_V to retrieve all the Generic Visualization Business Area instances on which a user has privileges to read data.

You can use this view to build a hierarchy in the visualization tool's user interface that displays each Business Area instance to which the current user has access, in the context of its Work Area, Application Area, and Domain.

This effectively enforces security by allowing the user to select only Business Areas to which he or she has access.

For each Business Area, the view retrieves the following:

COMPANY_ID. NUMBER(6,0) The Business Area instance's company ID.

BA_OBJ_ID. NUMBER(22,0) The Business Area instance's object ID.

BA_OBJ_VER. NUMBER(7,0) The Business Area instance's object version.

BA_NAME. VARCHAR2(200) The Business Area instance's name.

SCHEMA_NAME. VARCHAR2(30) The Business Area instance's unique schema name.

BA_DESCRIPTION. VARCHAR2(2000) The Business Area instance's description, if any.

BA_STATUS_RC. VARCHAR2(30) The Business Area instance's status. The possible values are: $NAMING_STATUS$UPGRADEABLE or $NAMING_STATUS$NONINSTALLABLE. See the *Oracle Life Sciences Data Hub Application Developer's Guide* for more information.

BA_VALIDATION_STATUS_RC. VARCHAR2(30) The Business Area instance's validation status. The possible values are: $SYSVALDNSTEPS$DEVELOPMENT, $SYSVALDNSTEPS$PRODUCTION, $SYSVALDNSTEPS$QUALITYCONTROL, $SYSVALDNSTEPS$RETIRED.

BA_VERSION_LABEL. VARCHAR2(255) The Business Area instance version's label, if any.

WA_OBJ_ID. NUMBER(22,0) The Business Area instance's parent Work Area's object ID.

WA_OBJ_VER. NUMBER(7,0) The Business Area instance's parent Work Area's object version.

WORKAREA. VARCHAR2(4000) The Business Area instance's parent Work Area's name.

WA_VALIDATION_STATUS_RC. VARCHAR2(30) The Business Area instance's parent Work Area's validation status. The possible values are: $SYSVALDNSTEPS$DEVELOPMENT, $SYSVALDNSTEPS$PRODUCTION, $SYSVALDNSTEPS$QUALITYCONTROL, $SYSVALDNSTEPS$RETIRED.

APPLICATION_AREA. VARCHAR2(4000) The Work Area's parent Application Area's name.

DOMAIN. VARCHAR2(4000) The Application Area's parent Domain's name.

Note:

Multiple levels of Domains are not included in the hierarchy.

BA_DEF_OBJ_ID. NUMBER(22,0) The Business Area definition's object ID.

BA_DEF_OBJ_VER. NUMBER(7,0) The Business Area definition's object version.

BA_DEF_NAME. VARCHAR2(200) The Business Area definition's name.

Retrieve Table Instance Details

Use view CDR_PUB_GENERIC_BA_TABLES_V to retrieve the Table instance details for a given Generic Visualization Business Area instance in order to determine what data to display.

The view retrieves the following attributes for each mapped Business Area Table Descriptor and Table instance. See the *Oracle Life Sciences Data Hub Application Developer's Guide* for more information.

COMPANY_ID. NUMBER(6,0) The Table instance's company ID.

BA_OBJ_ID. NUMBER(22,0) The Business Area instance's object ID.

BA_OBJ_VER. NUMBER(7,0) The Business Area instance's object version.

BA_NAME. VARCHAR2(200) The Business Area instance's name.

TD_OBJ_ID. NUMBER(22,0) The Table Descriptor's object ID.

TD_OBJ_VER. NUMBER(7,0) The Table Descriptor's object version.

TD_NAME. VARCHAR2(200) The Table Descriptor's name.

TD_ORACLE_NAME. VARCHAR2(30) The Table Descriptor's Oracle name.

TD_SAS_NAME. VARCHAR2(32) The Table Descriptor's SAS name.

TI_OBJ_ID. NUMBER(22,0) The Table instance's object ID.

TI_OBJ_VER. NUMBER(7,0) The Table instance's object version.

TI_NAME. VARCHAR2(200) The Table instance's name.

BLINDING_FLAG_RC. VARCHAR2(30) The Table instance's Blinding flag setting. The possible values are: `$YESNO$YES` if the Table instance can contain blinded data or `$YESNO$NO` if it cannot.

BLINDING_STATUS_RC. VARCHAR2(30) The Table instance's Blinding status. See the *Oracle Life Sciences Data Hub Application Developer's Guide* for more information.

- If the Table Instance's Blinding flag is set to `$YESNO$YES`, the possible values are: `$BLIND_STATS$BLINDED`or `$BLIND_STATS$UNBLINDED`.
- If the Table Instance's Blinding flag is set to `$YESNO$NO`, the possible values are: `$BLIND_STATS$NOTAPPLICABLE` or `$BLIND_STATS$AUTHORIZED`.

Generic Visualization Business Area Instance Attributes

Business Area instances of type Generic Visualization have several attributes including Schema Name, Default Currency, and Default Blinding Access Type.

For details, see the following:

- Schema Name
- Default Currency
- Default Blinding Access Type

Schema Name

The Definer must enter a unique name for the schema of up to 30 characters.

The value is stored in uppercase.

If a schema already exists in the Oracle Life Sciences Data Hub 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.

The first time the Definer installs the Business Area instance, the installation function creates a database schema with this name exclusively for the Business Area instance.

Default Currency

This setting controls the currency of data viewed by users if they do not explicitly make another selection.

The default value is Current. If all the source Table instances mapped to the Business Area have same snapshot label(s) applied, the Definer can select a shared snapshot for users to see by default.

Default Blinding Access Type

This setting controls the default blinding status of data available to users.

Users' privileges also determine which data they can view. The Definer's own privileges determine which values he or she can set. The possible settings include:

- **NA/Dummy** Allows users to see all 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**, and if the Definer has the Read(Unblind) Business Area Instance operation on the Business Area instance, the Definer 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. Users without these privileges must explicitly select the NA/Dummy blinding access type or else they will not be able to see data in the Business Area at all. All users can see all 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, regardless of the Default Blinding Access Type. All blind breaks are audited.

Generic Visualization Adapter Security

Security for the generic visualization adapter has several components including Database and User Accounts, Object Security, Data Blinding and Currency, and Auditing.

For details, see the following:

- Database and User Accounts
- Object Security
- Data Blinding and Currency
- Auditing

Database and User Accounts

You can use normal Oracle Life Sciences Data Hub security by assigning Oracle Life Sciences Hub user accounts to user groups and assigning user groups to Business Area instances, but if your users do not need Oracle Life Sciences Hub user accounts for other purposes, you can use simplified security requirements that apply only to Generic Visualization Business Area instances.

Users must log in using an Oracle Life Sciences Hub database account. Database accounts can have the following privileges directly assigned, either by the Business Area Definer with Manage BA DB privileges on the Business Area instance or by an Administrator:

- **Read Data** allows the user to see nonblinded and dummy data. All database accounts to be used to access a Generic Visualization Business Area intance should have this privilege.
- **Read Unblind** allows the user to see unblinded data.

Users who should be able to read data that was never blinded, dummy data in blinded Table instances and, optionally, data that has been unblinded, can log in using a database account that has the required privilege(s) directly assigned. They do not need an Oracle Life Sciences Hub user account.

Users who should be able to view currently blinded data must have their own Oracle Life Sciences Hub database account and a linked Oracle LSH user account with the privileges normally required for blind breaks, including:

- LSH Data Blind Break User application role
- Blind Break privileges on every Table instance whose Blinding Status is Blinded and that is mapped to one of the Business Area instance's Table Descriptors
- The user account must be assigned to a user group that is assigned to the Business Area instance

When a user logs in, which requires an Oracle Life Sciences Hub database account, the Initialization API checks for a linked Oracle Life Sciences Hub user account. If there is one, the API uses that account's privileges to determine what data the user can view. If there is no linked Oracle Life Sciences Hub user account, the user has access only to the data to which the database account has access.

To assign privileges to database accounts in the Business Area instance user interface itself, the Definer selects **Manage DB Privileges** from the **Actions** drop-down. The security administrator can do the same for any Business Area instance in the **Security** user interface, **BA DB Privilege Access** tab. All the database accounts defined in the Oracle Life Sciences Hub instance are available for assignment.

See the *Oracle Life Sciences Data Hub System Administrator's Guide* for information on creating Oracle Life Sciences Hub user and database accounts and the *Oracle Life Sciences Data Hub Implementation Guide* for an explanation of Oracle Life Sciences Hub security.

Object Security

To view data, users must have privileges on a particular Business Area instance.

You can use a view to display for selection only the Business Area instances to which the current user has access; see Display User's Business Area Instances.

During installation the system creates synonyms in the Business Area instance schema for all the Table Descriptors. The synonyms reference source views in the Work Area schema(s) containing the actual database tables corresponding to the Table instances mapped to the Table Descriptors. The source views enforce normal Oracle LSH security.

Note:

It is not necessary to have privileges on the Table instances mapped to the Business Area instance's Table Descriptors unless the user needs to be able to view currently blinded data.

Data Blinding and Currency

You can use APIs to determine the appropriate blinding and currency settings for the current session, based on the state of the data and the user's privileges, and allow the user to select only those settings.

See Getting Possible Blinding Types of a Business Area Instance and Getting Snapshot Labels Common to All Tables in a BA Instance for a Given Blinding Access Type.

Auditing

The adapter audits several actions, recording each in internal tables as follows.

You can log in as `apps` to see the audit trail for each.

- CDR_BA_SCHEMA_ACCESS stores a record for each user access to a Business Area instance schema using the initialization API.
- CDR_BLIND_BREAKS stores a record of each blind breaks (sessions where a privileged user views currently blinded data).
- CDR_BA_DB_PRIVILEGES_A is the FND audit table for the CDR_BA_DB_PRIVILEGES, which stores the association between database accounts and Generic Visualization adapter privileges.

Generic Visualization Adapter Definitional Components

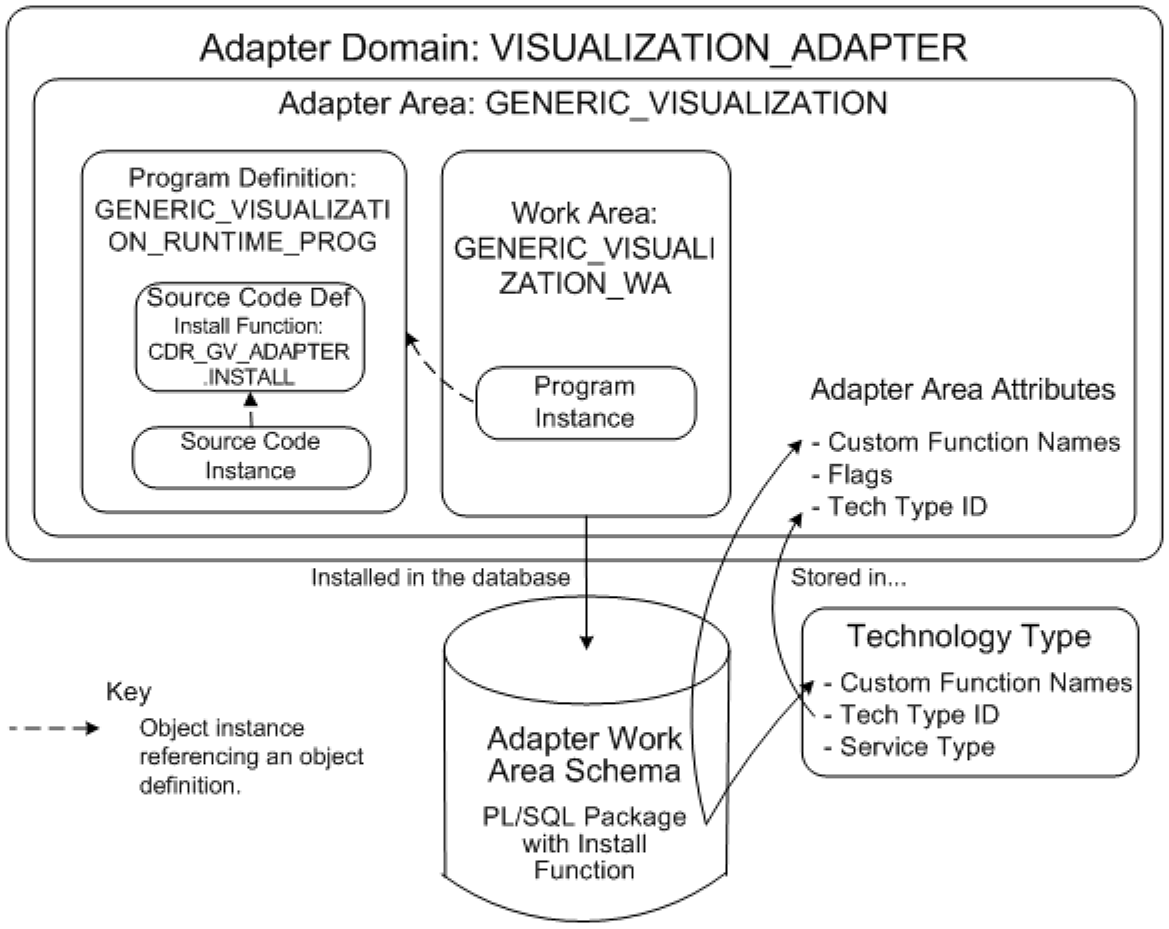
All the defined objects, from the Adapter Domain down, are shipped with Oracle Life Sciences Data Hub.

As shown in Figure 5-1 these include:

- **Adapter Domain** VISUALIZATION_ADAPTER
- **Adapter Area** GENERIC_VISUALIZATION; see Table 4-2 for its attribute values.
- **Program** GENERIC_VISUALIZATION_RUNTIME_PROG contains a Source Code definition that includes a single function, the install_function, named CDR_GV_ADAPTER.INSTALL. This function is called whenever a Definer installs a Generic Visualization Business Area instance. It does the following:
  - If a schema does not exist for the Business Area instance, it creates the schema.
  - For each Table Descriptor in the Generic Visualization Business Area, it creates or upgrades (as required) a corresponding synonym in the schema that references a source view of the mapped Table instance in its Work Area schema.
  - If a previously created Business Area instance had a schema with the same name, but that Business Area instance has been deleted, the install function drops all the synonyms in the original schema and creates new ones for the new Business Area.
- **Work Area** GENERIC_VISUALIZATION_WA containing an instance of the Program GENERIC_VISUALIZATION_RUNTIME_PROG. The Adapter Area Work Area is installed in its own Oracle Life Sciences Data Hub schema.
- **Technology Type** GVA; see Table 4-1 for its attribute values.

> **Note:**
>
> Generic Visualization Business Area instance attributes Schema Name, Default Currency, and Default Blinding Access Type are not stored in the define-time Parameter Set as in other adapters. They are stored in satellite table CDR_BUSINESS_AREA_REFS so that the public APIs can change their values.

Figure 5-1 Generic Visualization Adapter Components


Adapter Domain: VISUALIZATION_ADAPTER
Adapter Area: GENERIC_VISUALIZATION
Program Definition: GENERIC_VISUALIZATION_RUNTIME_PROG
Source Code Def Install Function: CDR_GV_ADAPTER.INSTALL
Source Code Instance
Work Area: GENERIC_VISUALIZATION_WA
Program Instance
Adapter Area Attributes
- Custom Function Names
- Flags
- Tech Type ID
Installed in the database
Stored in...
Technology Type
- Custom Function Names
- Tech Type ID
- Service Type
Key
Object instance referencing an object definition.
Adapter Work Area Schema
PL/SQL Package with Install Function

6 Checking In Objects and Setting Their Validation Status

Checking in all objects is required. You can also set the validation status.

Setting objects' validation status is optional. It does not change any behavior and the status is not visible in the user interface or to end users in any way. However, it may be useful for your internal testing and validation purposes to use this tool. In addition, the API that upgrades objects' validation status first checks in all objects and can be used to simplify that task.

The validation API checks in the object you specify in the input parameter PI_VALOBJ, and all the objects it contains (if any), and upgrades its validation status. If you specify the Adapter Domain in PI_VALOBJ, the API attempts to check in and upgrade all the objects in your adapter.

Note:

The following conditions prevent a successful validation:

- The validation of a particular object fails if the object is checked out by a user different from the person running the validation API. In this case, the person who has the object checked out must either explicitly check in the object or run the validation API.
- The validation process fails if the Work Area is included in the process (which it is if you have specified either the Adapter Domain, the Adapter Area, or the Work Area itself) and does not have a validation status of Production. Use the API CDR_PUB_DF_WORKAREA.UPDATEUSAGEINTENT to change it to Production.

For more details, see the following:

- Running the Validation API
- Explicitly Checking In an Object

Running the Validation API

To set the validation status of all the objects in your Adapter Domain to Development, Quality Control, or Production, run the API CDR_PUB_VL_VALIDATION.UPDATEVALSTATUS and provide values relating to the Adapter Domain.

Enter mandatory values or parameter PI_VALOBJ to specify your Adapter Domain:

- **company_id** = `Enter_your_company_ID`

- **obj_id** = `Enter_your_Adapter_Domain_ID`
- **obj_ver** = 1
- **obj_type_rc** = `'$OBJTYPES$ADAPTERDOMAIN'`
- **validation_status_rc** = Enter one of the following values: `$SYSVALDNSTEPS$RETIRED, $SYSVALDNSTEPS$DEVELOPMENT, $SYSVALDNSTEPS$QUALITYCONTROL, $SYSVALDNSTEPS$PRODUCTION`
- **object)_version_number** = 1

Two output parameters indicate which objects were and were not successfully upgraded:

- PO_CASCADEDOBJCOL lists the objects that were successfully upgraded
- PO_ERRORNAMINGCOLL lists the objects that could not be upgraded because they are checked out by another user

Explicitly Checking In an Object

There is a different API for checking in each type of object.

You have already explicitly checked in Variables and Parameter definitions (see Check in the Variable and Check in the Parameter.

You may need the following APIs:

- CDR_PUB_DF_PROGRAM.CHECKINPROGRAMDEFINITION
- CDR_PUB_DF_PARAMETER_SET.CHECKINPARAMETERSETDEFINITION
- CDR_PUB_DF_TABLE.CHECKIN

Each of these APIs has the same type of parameters:

- PIO_BASEOBJECT. Enter CDR_BASE_OBJ_TYPE values to identify the object.
  - **company_id** Enter your company ID; see Getting Your Company ID.
  - **obj_id** Enter the object's Object ID.
  - **obj_ver** = 1
  - **object_version_number** = 1
  - **namespace_obj_id** Enter your Adapter Area's Object ID.
  - **namespace_obj_ver** = 1
- PI_COMMENT(Optional) Enter the reason you are checking in the Program.

7 Setting Up an Adapter

In order to use and test your adapter by creating and running an object of the relevant type, you must do certain tasks in your local Oracle Life Sciences Data Hub installation. If you are developing an adapter for use in other companies, provide information about these tasks to the other companies.

For details, see the following:

- Defining a Service Location, Service, and Service Instance
- Defining Remote Locations and Connections
- Assigning User Groups to the Adapter Area
- Installing and Starting the Distributed Processing Server
- Edit cdrconfig.xml

Defining a Service Location, Service, and Service Instance

Do the following tasks in the Oracle Life Sciences Data Hub user interface.

Instructions are in the *Oracle LSH System Administrator's Guide*.

- Add the service types required for the adapter to the Lookup CDR_SERVICE_TYPES.
- Define a service location for the computer where the external system is installed.
- Define at least one service for each service type you added to the lookup.
- Define at least one service instance for each service.

Defining Remote Locations and Connections

For Oracle-based systems, define at least one Remote Location and Connection.

See the chapter on registering remote locations and connections in the *Oracle LSH System Administrator's Guide*.

Assigning User Groups to the Adapter Area

Assign one or more user groups to your Adapter Area.

Users in user groups assigned to the Adapter Area can see and select the new adapter type when they create a new Load Set, Data Mart, Program, or Business Area. If you have not already created user groups, see the *Oracle LSH System Administrator's Guide* for more information.

Note:

The users in user groups assigned to Adapter Areas should have roles that include **ONLY View privileges**. Giving users Modify privileges on the objects in the Adapter Area could result in the Adapter's becoming invalid and no longer working.
For example, if a user modifies a predefined Parameter in a Load Set or Data Mart, the system automatically creates a new version of the Parameter definition in the Adapter Area, creating a new version of the Adapter Area itself, and the adapter becomes invalid.

You can assign user groups to Adapters in the user interface or by running an API.

User Interface Method

User interface instructions are included in the chapter on setting up adapters in the *Oracle LSH System Administrator's Guide*.

API Method

Run the API CDR_PUB_SECURITY_PKG.ASSIGNUSRGRPTOOBJ. See API Method.

Installing and Starting the Distributed Processing Server

For adapters that use the Distributed Processing (DP) Server during execution (those that run on the operating system rather than PL/SQL), install and start the Distributed Processing (DP) Server on the computer where the external system is installed.

Instructions for installing the DP Server are in the *Oracle LSH Installation Guide*; instructions for starting the DP Server are in the *Oracle LSH System Administrator's Guide*.

Edit cdrconfig.xml

The cdrconfig.xml shipped in the Oracle Life Sciences Data Hub client plug-in, cdrclient.exe, includes a section for each adapter.

You must add a section to this file for your adapter that includes the path to the installed external system; see Planning Integrated Development Environment Adapters.

At each installation, each Definer must copy this file to his or her computer, edited to include the actual path to the system.

If you are developing an adapter for use by other companies, provide instructions for editing the file.

See the *Oracle LSH Installation Guide* Section 7.15.2.1 "Install the Client Plug-In" for further information.

8

Shipping an Adapter

To package an adapter so that you can ship it for use in other companies, you must provide an installation script and instructions for your customers.

For details, see the following:

- Installation Script
- Instructions

Installation Script

Write a PL/SQL installation script that loads both of the packages you created in Step 1 of development—with the final, production-quality functions and procedures.

The installation script must:

- Create an input parameter that accepts the value of the user group ID of the user group the customer company wants to assign to the Adapter Area.
- Read Package B from the file system, create a CLOB of the file, and pass the CLOB to the CreateSourceCode API in Package A; see Adapter Development Process.

Instructions

Provide instructions that include the following information:

- The user group to be assigned to the Adapter Area should include a role with View (only) access to the Adapter Area and all objects within it, and should include all users who will need to define or run Load Sets, Data Marts, Programs, or Business Areas of the new adapter type.
  The user who is running the installation script must also be included in the user group with a role that has Create and Modify privileges on all objects in the Adapter Area.
- Customers must follow all the steps in Setting Up an Adapter.