## Contents

Preface

1 Modeling and Sharing Solution Data

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>About Data Schemas</strong></td>
<td>1-1</td>
</tr>
<tr>
<td><strong>About Data Elements</strong></td>
<td>1-1</td>
</tr>
<tr>
<td>About Primitive Data Types</td>
<td>1-2</td>
</tr>
<tr>
<td>About Data Element Icons</td>
<td>1-3</td>
</tr>
<tr>
<td>About the Data Modeling Tabs</td>
<td>1-3</td>
</tr>
<tr>
<td>About the Details Tab</td>
<td>1-3</td>
</tr>
<tr>
<td>About the Enumerations Tab</td>
<td>1-4</td>
</tr>
<tr>
<td>About the Tags Tab</td>
<td>1-5</td>
</tr>
<tr>
<td>About the Usage Tab</td>
<td>1-5</td>
</tr>
<tr>
<td>About the Information Tab</td>
<td>1-6</td>
</tr>
<tr>
<td>Data Element Application Details</td>
<td>1-7</td>
</tr>
</tbody>
</table>

**About Data Modeling Strategies and Techniques**                      | 1-7  |

Leveraging Information from Existing Data Elements                     | 1-7  |
Deriving from Base Type Elements                                       | 1-7  |
Aliased Data Elements                                                  | 1-8  |
Data Element Recursion                                                 | 1-9  |
Entity Extensions                                                       | 1-11 |
Inheritance                                                            | 1-12 |
Organizing and Searching for Data Elements                             | 1-13 |
Data Model Hierarchies                                                 | 1-13 |
Data Model Organization                                                | 1-14 |
Data Element View                                                       | 1-15 |
Navigation and Modeling Tips                                           | 1-15 |
Refactoring Data Models                                                | 1-17 |
Renaming Data Elements                                                 | 1-17 |
Removing Data Elements                                                 | 1-17 |
Making Data Elements Reusable                                          | 1-18 |
Replacing Data Elements                                                | 1-19 |

**Working with Externally Created Schemas**                            | 1-19 |
About Design Studio Data Schemas                                       | 1-19 |
Modeling Data Using XML Data Schemas                                   | 1-20 |
About Supported XML Schema Features                                    | 1-20 |
About Unsupported Schema Directives and Elements                        | 1-21 |
## 2 Working with Design Patterns and Guided Assistance

About the Design Pattern and Guided Assistance SDK Folder .................................................. 2-1

**Working with Design Patterns** ....................................................................................... 2-2

- About Design Patterns ....................................................................................................... 2-2
- Creating Design Patterns .................................................................................................... 2-3
- Modeling Design Pattern Content ...................................................................................... 2-4
  - About Design Pattern Configuration Data ...................................................................... 2-4
  - About Design Pattern Resources ..................................................................................... 2-4
- Working with Tokens .......................................................................................................... 2-5
  - Token Types ...................................................................................................................... 2-5
  - Defining Tokens ............................................................................................................... 2-6
  - Examples: Defining Tokens ............................................................................................. 2-6
  - Embedding Tokens as Default Values ............................................................................. 2-7
  - Restricting Token Values .................................................................................................. 2-7
  - About Element References ............................................................................................. 2-7
- Testing Design Patterns ........................................................................................................ 2-8
- Securing Design Pattern Information .................................................................................. 2-8
- Distributing Design Patterns ............................................................................................... 2-8
- Applying Design Patterns .................................................................................................... 2-9

**Working with Guided Assistance** .................................................................................... 2-10

- About the Guided Assistance Dialog Box ........................................................................... 2-10
- Working with Guided Assistance Design Patterns ............................................................. 2-10
- Creating Guided Assistance Using Design Patterns ......................................................... 2-10
- Working with the Guided Assistance Extension Point ......................................................... 2-12
  - Guided Assistance Extension Point Example .................................................................. 2-13
- Distributing Guided Assistance .......................................................................................... 2-14

## 3 Working with Source Control

About Source Control ............................................................................................................ 3-1

- About Source Control Strategies for Design Studio Files .................................................. 3-1

## 4 Deploying Cartridges to Environments

- Deploying Cartridges to Run-Time Environments with the Cartridge Management Tools Utility 4-1

- Working with Additional Cartridge Deployment Tools ....................................................... 4-2

## 5 Design Studio Platform Tools

- Working with Oracle Enterprise Packet for Eclipse .......................................................... 5-1
  - About Java Development Tools ....................................................................................... 5-1
  - About Database Development Tools ............................................................................... 5-2
  - About Application and WebLogic Server Tools .............................................................. 5-2
  - About Web Application Tools ......................................................................................... 5-2
  - About JPA and Oracle Coherence Tools ......................................................................... 5-3
- About Third-Party Tools ..................................................................................................... 5-4
Preface

This guide explains how to work with Oracle Communications Design Studio design patterns, guided assistance, and externally created schemas. It explains how to automate builds and provides information about implementing continuous integration to design, create, and deliver operations support system (OSS) solutions across Oracle Communications products.

This guide assumes that you have a conceptual understanding of Design Studio and have read Design Studio Concepts.

This guide includes examples of typical development code used in given situations. The guidelines and examples may not be applicable in every situation.

Audience

This guide is intended for developers who work with Design Studio to build the code to support the metadata-driven components of Design Studio projects. You should have a good working knowledge of development languages such as Java, XPath, XQuery, or SQL create service fulfillment solutions.

Related Documents

For more information, see the following documents in the Design Studio documentation set:

- **Design Studio Installation Guide**: Describes the requirements and procedures for installing Design Studio.

- **Design Studio Concepts**: Explains how to use Design Studio to manage and configure data for use across Oracle Communications service fulfillment products. This guide provides a conceptual understanding of Design Studio.

- **Design Studio System Administrator’s Guide**: Describes information about administering Design Studio. This guide includes information about configuring deployment settings for test environments, backing up and restoring Design Studio data, and automating builds.

- **Design Studio Security Guide**: Provides an overview of security considerations, information about performing a secure installation, and information about implementing security measures in Design Studio.

This chapter describes the features and functionality you use in Oracle Communications Design Studio to model solutions for and share data across service fulfillment and network and resource management business solutions.

About Data Schemas
A data schema is an XML schema that provides a formal description of a data model, expressed in terms of constraints and data types governing the content of elements and attributes.

You use data schemas when defining products, services, and resources, including the associated actions and the information necessary to perform the processes and tasks for those actions, as well as the interface definitions for integrating between applications.

All data elements are created and saved in data schemas, which can be accessible across all projects in a workspace. Design Studio automatically creates a project-specific data schema when you create a cartridge project (for example, an OSM, Activation, Inventory, or Network Integrity project). You can use this default schema to contain the data you require to model the project, you can create multiple schemas in the same project, or you can create schemas in common projects. You can model your cartridge project using data from any combination of these data schemas.

Model project data schemas include data elements that you want to use across a fulfillment solution. They are product-agnostic; that is, the data elements stored in a model schema are independent of any application project.

About Data Elements
When modeling data for a project, you can create simple and structured data elements that you can reuse throughout your model. Simple data elements are reusable data types that contain no child dependencies. A simple data element has no structure, and is associated—directly or indirectly—to a primitive type (Integer, Boolean, String, and so forth). See "About Primitive Data Types" for more information.

Structured data elements are reusable data types that include embedded data types and are containers of simple data elements and other structured data elements. For example, you might create a structured data element called building that contains the floor, room, aisle, rack, and shelf child structured data elements.
About Primitive Data Types

There are many primitive types of simple data elements, such as Boolean, Decimal, and String. Design Studio supports primitive types defined by the XML Schema specification.

Table 1–1 lists the full set of supported types and their formats. These types represent the foundation from which all data elements in Design Studio are derived.

<table>
<thead>
<tr>
<th>Type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Hierarchical construct of structures and simple data elements.</td>
</tr>
<tr>
<td>Boolean</td>
<td>true or false</td>
</tr>
<tr>
<td>Date</td>
<td>yyyy-mm-dd</td>
</tr>
<tr>
<td></td>
<td>For example: 1971-05-21</td>
</tr>
<tr>
<td>Date Time</td>
<td>yyyy-mm-dd-ddThh:mm:ss.szzzzzz</td>
</tr>
<tr>
<td></td>
<td>where zzzzzz represents a time zone, s represents fractional seconds</td>
</tr>
<tr>
<td></td>
<td>Fractional seconds and time zone are optional.</td>
</tr>
<tr>
<td></td>
<td>Use Z for Coordinated Universal Time (UTC), or UTC relative form (+</td>
</tr>
<tr>
<td>Time</td>
<td>hh:mm:ss</td>
</tr>
<tr>
<td></td>
<td>For example: 14:22:35</td>
</tr>
<tr>
<td>Integer</td>
<td>Infinite set {..., -2, -1, 0, 1, 2, ...} using decimal digits</td>
</tr>
<tr>
<td>Long</td>
<td>between -9223372036854775808 and 9223372036854775807, inclusive</td>
</tr>
<tr>
<td>Float</td>
<td>Lexical representation consisting of a significand and an optional exponent.</td>
</tr>
<tr>
<td>Double</td>
<td>For example: -1E4</td>
</tr>
<tr>
<td>Heximal</td>
<td>1267.43233E12</td>
</tr>
<tr>
<td></td>
<td>12.78e-2</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Hex Binary</td>
<td>[0-9a-fA-F] tuples</td>
</tr>
<tr>
<td></td>
<td>Must have an even number of hexadecimal digits. Hex binary allows length restriction.</td>
</tr>
<tr>
<td></td>
<td>For example: A1, 80FF, 18A2C797</td>
</tr>
<tr>
<td>String</td>
<td>Series of UTF-8 characters.</td>
</tr>
<tr>
<td></td>
<td>String allows length restriction (default 0 to 40).</td>
</tr>
</tbody>
</table>
About Data Element Icons

Design Studio communicates various properties of data elements by associating icons with the data elements. These icons include problem marker severity, data element multiplicity, and some state details. Additionally, data elements that are read-only appear in gray.

**Note:** The information conveyed through these icons is also available on the Details tab. The problem marker information is available in the Problems view. See Design Studio Help for more information about the Details tab fields and for more information about the Problems view.

Figure 1–1 displays a list of icons and the manner in which they appear with data elements.

*Figure 1–1 Design Studio Data Element Icons*

<table>
<thead>
<tr>
<th>Problem Marker</th>
<th>Description</th>
<th>Structured</th>
<th>Simple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error marker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warning marker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deprecated</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inherited</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aliased</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

About the Data Modeling Tabs

Some Design Studio editors include common tabs in which you can model information about data elements. These tabs appear in multiple Oracle Communications applications and enable you to configure entities by modeling a data tree to hierarchically represent all associated data elements. These common tabs facilitate reusing data elements within a modeling solution and provide tools for locating and using existing data elements.

These common tabs consist of a data tree and subtabs. You select a data element in the data tree to review and model details of the selected data element. You open a context menu from the data tree to perform various types of refactoring operations on the data elements.

About the Details Tab

You use the Details tab to define attributes of a data element (such as the name and primitive type), as well as specific constraint values for a data element.
About the Enumerations Tab

You use the **Enumerations** tab to define sets of valid values for data elements. Enumerations define values for data elements that are available for selection in a run-time environment. For example, you can define a set of values that appear as lists in the run-time environment.

**Figure 1–3  Enumerations Tab**

- Enumerations inherited from a base type can be excluded
- Descriptions can be specified in multiple languages
- Enumerations can be added and removed

- **Include**
  - Code: red
  - Description: Red Widget
  - Def.: Yes
  - Language: [default]
- **Exclude**
  - Code: green
  - Description: Green Widget
About the Tags Tab
You use the Tags tab to characterize data elements by associating the data elements with predefined keywords called tags. Tags help you filter and search for data elements that are associated with specific Oracle Communications applications.

The tags delivered with Design Studio cannot be inherited.

*Figure 1–4 Tags Tab*

About the Usage Tab
You use the Usage tab to review the projects and entities in which a data element is used and to review all references to a specified data element.
Figure 1–5  Usage Tab

About the Information Tab
You use the Information tab to annotate data elements with descriptions or other applicable information that support the data element.

Design Studio supports multiple languages for this tab. The field at the top of this tab displays your list of languages. If your preferences are set up to work in one language only, the system displays only [default].

Figure 1–6  Information Tab
Data Element Application Details
You can define for data elements application-specific properties, which you configure using the application-specific data element tabs. For example, you can define information for data elements that are tagged as Inventory characteristics, or you can define behaviors for data elements that will appear in an OSM run-time environment.

Figure 1–7 Application-Specific Tabs

About Data Modeling Strategies and Techniques
Design Studio enables you to:

■ Create new data elements that obtain attributes from other data elements. See "Leveraging Information from Existing Data Elements" for more information.

■ Use components and features to search for and organize data elements. See "Organizing and Searching for Data Elements" for more information.

■ Use the refactoring tools to normalize data models. See "Refactoring Data Models" for more information.

Leveraging Information from Existing Data Elements
To increase modeling efficiency, Design Studio enables you to create new data elements that obtain attributes from other data elements. In Design Studio, you can leverage information using two methods:

■ Deriving from a base type element (where the new element automatically obtains the information in the base element

■ Extending entities

Deriving from Base Type Elements
Leveraging information already defined for base types enables you to define attributes once, share the common attributes among multiple entities, and edit those entities in a single location. Changes that you make to a base type are automatically saved to all entities that derive from that base type.

For example, if a structured data element person contains the first_name, last_name, and social_security_number child elements, you can leverage the information defined
for **person** by using it as a base type for a new structured data element called **employee**. You can add to **employee** the **employee_number**, **hire_date**, and **department** child elements. If you make changes to **person**, Design Studio automatically updates **employee** with the same changes.

You can override some of the information derived from the base type element. When you override these fields, Design Studio does not automatically update that value when the base type value is changed. Rather, Design Studio retains the override value that you defined for the subtype field.

Attributes that you can override include the minimum and maximum number of times the element can appear as a field in a run-time environment, the default value defined for the element, the length defined for string and HexBinary data elements, internal documentation for the element, and so forth.

**Figure 1–8  Derived Data Elements**

**Aliased Data Elements** Typically, derived data elements inherit the name of their base type. If you want to give a derived data element a unique name, you can define an alias for the data element. You can use an alias to give context to the specific use of a data element or to distinguish multiple data elements derived from the same base type.
Data Element Recursion  Data element recursion refers to a pattern where a data element repeats in the ancestry of derived data elements. Recursion provides useful patterns for data modeling, and Figure 1–10 demonstrates some of these patterns. When recursion is present, the point of recursion starts an infinite nesting of inherited data elements. Most variants of structured data element recursion are valid in Design Studio.

Simple data element recursion, however, is not valid in Design Studio models because a data element can never resolve to a primitive type. Design Studio validates base type selection to prevent you from defining simple data elements as recursive. If an invalid recursion occurs, Design Studio generates an error marker during the build process.
Structure data elements with recursion inherit the data elements of the base type. The Design Studio user interface expands to display a finite number of recursion levels (the number of levels is configurable in the workspace preferences). All structured data elements, including derived structured data elements, can be extended to include additional child data elements.

A recursive reference can be indirect when two structures have data elements that derive from each other. Indirect relationships can be realized through multiple levels of derivation and are not immediately apparent. Figure 1–11 illustrates two different data models using recursion. Figure 1–12 is a representation of these models and provides an example of a recursive structured data element.
Entity Extensions

You can increase your modeling efficiency by extending entities. Data extensibility enables you to leverage data when building new, similar entities. For example, you can extend orders and tasks. When you extend one entity from another, the target entity inherits all of the data elements defined for the extended entity.
If you extend an entity that includes structured data elements, you can add any number of additional simple and structured child elements (the inherited data elements are read-only).

Inheritance When you extend one entity from another, the target entity inherits all of the data elements defined for the extended entity. These inherited data elements are read-only. Unlike data elements inherited from a base type, the inherited data elements can be extended at any level of the inherited data element.

Figure 1–14 demonstrates how an OSM manual task extends another manual task. The data elements modeled on one manual task are extended to another manual task. The example includes additional structures added to the extended manual task, illustrating how content can be extended.
Organizing and Searching for Data Elements

Design Studio helps you keep your data models organized and locate specific data elements during model configuration.

Data Model Hierarchies

Figure 1–15 defines the categories of projects in which you can model data schema entities. Understanding to which category a project belongs will help keep your model organized and efficient. When determining the location for data elements, consider how specific that data element is to a product or to a domain.

For example, common product projects are often sealed models provided by the product teams for solution development, whereas domain-specific model projects are ideal for domain-specific content.

Oracle recommends that you always model in the most generic category possible.
Data Model Organization
You can model data elements in various locations in a solution. Figure 1–16 defines the different categories of projects based on their specificity to a domain or product. Arrows indicate best practice dependencies between types of modeling components. Shaded arrows indicate preferred dependencies because they promote the best cross-product and cross-domain reuse.
Data Element View

You can use the Data Element view to facilitate data modeling. The Data Element view presents a view of the Data Dictionary, including all reusable data elements contributed by entities in the workspace.

The Data Element view includes an option to filter data elements visible only to the active editor, which enables you to hide those elements in the Data Dictionary that are not intended to be used by specific entities. Additionally, you can filter this view to display only those entities flagged with a specific tag or defined by a specific entity type.

Search fields provide an additional mechanism to locate data elements.

Figure 1–17 Data Element View

Navigation and Modeling Tips

Design Studio includes shortcuts for navigation. When switching between views and editors, these shortcuts help minimize modeling time and effort.

For example, you can double-click any editor or view tab to maximize the editor or view area. Double-click the tab again to return it to the original size. From the Data Element view, double-clicking a data element opens the related entity. Double-clicking a data element in an editor opens the base type data element. Clicking the Type field link in the Data Schema editor Details tab opens the base type of an element.

Figure 1–18 illustrates some areas in the Design Studio user interface where you can use these shortcuts and navigation tips.
You can drag data elements from the Data Element view to an editor data tree when modeling your solution.

Figure 1–19 Dragging Data Elements to Editors
Refactoring Data Models

In Design Studio, refactoring is the process of changing data elements without modifying the external functional behavior of a solution.

Refactoring in Design Studio enables you to propagate data model changes across the entire solution without sacrificing model integrity. You can rename, change the location of, copy, and move data elements. Additionally, refactoring enables you to copy data elements to create similar data entities, and to create modular and reusable data structures. See the Design Studio Help for more information.

The following sections include additional information about some of the refactoring actions, all of which are available from the **Refactoring** option in content menus.

**Renaming Data Elements**

The rename refactoring option ensures that all references are properly updated when you rename a data element, and that all aliased names are maintained.

Design Studio displays a detailed list of all the changes to be performed against the workspace that you can review.

In Figure 1–20, a data element named Caller ID is renamed to Call Display. The Rename refactoring functionality ensures that the entities that reference the data element (in this example, an atomic action called Add GSM Subscriber, an OSM order called Mobile GSM Delivery, and an Inventory supplementary service called Supplementary Service) are updated with the new data element name, Call Display.

**Figure 1–20 Refactoring Solutions Using Renaming**

![Refactoring Solutions Using Renaming](image)

**Removing Data Elements**

When you remove data elements from the workspace, Design Studio locates and removes all referenced instances. When you remove data elements, ensure that other constructs in the solution are updated appropriately.

In Figure 1–21, a data element named Anonymous Call Reject is deleted. The Design Studio refactoring functionality ensures that the data element is also deleted from all
entities that reference the data element (in this example, an atomic action, an OSM order, and an Inventory supplementary service).

**Figure 1–21  Refactoring Solutions Using Remove**

Making Data Elements Reusable
When a structure nested inside a structure is identified as a data element suitable for reuse, you can convert that structure into a reusable root level structure.

Often, the action to make an element reusable is combined with an action to move the data element to a generic location, such as a common data schema entity. In **Figure 1–22**, the Subscriber structure will be converted into a reusable data element. The original structure is a new, derived structure of the same hierarchy.

**Figure 1–22  Refactoring using Make Reusable**
Replacing Data Elements

Figure 1–23 demonstrates how a data element can be replaced with another compatible data element to normalize the solution data model, or to migrate to a newer representation. Depending on the purpose of the replacement, the original data element may be deprecated or removed after the replacement is complete.

When you refactor solutions using the replace and resolve actions, Design Studio ensures that suitable substitutions are made and that the overall model integrity is maintained. For example, data element replacements must not break any data element references or introduce primitive type inconsistencies. Replacements must meet the following criteria:

- Primitive types must match
- Structure is a match or superset
- Children of structures must be compatible
- Proposed data elements are reusable
- Replacement data element does not create an invalid recursion
- Replacement data element does not reference the data element being replaced
- Resolve action proposals are less specific

Working with Externally Created Schemas

Design Studio uses XML data schema for domain modeling. XML data schemas store data definitions and standardize data usage across domains and platforms.

About Design Studio Data Schemas

You create an XML data schemas by either creating a model project or by creating a schema in an application project. Design Studio saves the data schema entity to the
root level of the project’s dataDictionary directory by default. The Data Schema entity is represented by an XSD file and a companion file in the local file system.

You can also import any XML data schema file that was not created in a Design Studio project. If you create an XML data schema externally and edit it in Design Studio, you may lose data at the schema level and in the data model.

**Modeling Data Using XML Data Schemas**

Design Studio uses XML data schemas for data modeling. XML data schemas provide precise descriptions of data models that are bound by strict sets of rules, and they generate entities and associated features of a model. XML data schemas also support domain-specific requirements, such as inheritance and abstraction.

Design Studio simplifies using XML data schemas with the Data Dictionary. The Data Dictionary is a logical collection of data elements within the workspace and is presented within a set of views. These views enable you to visualize and manage the data elements configured in the workspace.


The data schema is represented in Design Studio as two files: an XML schema file and a companion file. Design Studio uses the companion file to save payloads that are declared in a data schema. For example, a provisioning system may require that all root level elements have cardinality. However, XML schema does not support cardinality for root level-type definitions. Design Studio saves this information in the data schema companion file. The companion file is hidden and is not visible in the Studio perspective.

**About Supported XML Schema Features**

Design Studio data schemas support a subset of the features of the XML Schema language. Some of the supported features are enhanced to optimize their use when modeling data.

Table 1–2 lists the XML Schema features that are supported.

<table>
<thead>
<tr>
<th>Feature (Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type Declaration</td>
<td>Supports the same definitions as the XML Schema specification.</td>
</tr>
<tr>
<td>Target Name Space</td>
<td>Supports the same definitions as the XML Schema specification.</td>
</tr>
<tr>
<td>Complex Content Type Definitions</td>
<td>Supports child structures and complex types.</td>
</tr>
<tr>
<td>Import Directives</td>
<td>Supports the same definitions as the XML Schema specification.</td>
</tr>
</tbody>
</table>
Table 1–2  (Cont.) Supported XML Schema Features

<table>
<thead>
<tr>
<th>Feature (Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardinality/Occurrence</td>
<td>Supports the cardinality/occurrence of the elements (child simple element from the XML Schema perspective). Also supports the cardinality on the type definitions on the root level, as well as child complex elements. Supports the various modeling needs of provisioning systems.</td>
</tr>
<tr>
<td>Max Length</td>
<td>Supports maximum length facets of XML Schema. All rules of maximum length facets apply.</td>
</tr>
<tr>
<td>Min Length</td>
<td>Supports minimum length facets of XML Schema.</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Supports the enumeration feature of the XML Schema specification, and enumerations for non-string elements. Fulfills the requirements of the various modeling needs of provisioning and inventory systems. Derived elements can extend the base enumerations, or exclude (restrict) them.</td>
</tr>
<tr>
<td>Annotations</td>
<td>Supports annotations on the elements and type definitions.</td>
</tr>
<tr>
<td>Deriving types by extension</td>
<td>Supports type definitions extended from other type definitions. While you can model recursive structures in Design Studio, the system restricts the presentation of recursive structures. In Design Studio, a data schema is represented as a hierarchical tree. Design Studio does not allow infinite expansion of recursive tree nodes. You can limit the number of levels to which nodes are expanded. See Design Studio Help for information about defining preferences.</td>
</tr>
</tbody>
</table>
| Primitive Data Types   | Supports the following primitive data types:  
  - int  
  - string  
  - long  
  - float  
  - double  
  - date  
  - date and time  
  - time  
  - boolean  
  - decimal  
  - hexadecimal |

About Unsupported Schema Directives and Elements

Design Studio data schemas support a subset of the features of the XML Schema language.

Table 1–3 lists the XML Schema features that are not supported.
### Table 1–3  Unsupported Schema Directives and Elements

<table>
<thead>
<tr>
<th>Schema Directives and Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include Directives</td>
<td>Valid external schemas that are imported into Design Studio and have an include directive may cause unresolved type definitions and can be considered invalid by Design Studio validation framework.</td>
</tr>
<tr>
<td>No target namespace</td>
<td>Schemas that have no Target Name space defined are not supported.</td>
</tr>
<tr>
<td>Redefine</td>
<td>If an external schema is using the Redefine element, the validation may result in unresolved elements.</td>
</tr>
<tr>
<td>Abstract Element and Types</td>
<td>If an external schema contains an Abstract definition, Design Studio considers it a regular element and does force a substitution according to the XML Schema specification.</td>
</tr>
<tr>
<td>Attribute Declaration and Attributes Groups</td>
<td>If an external schema contains an Attribute Declaration, Design Studio considers it to be read-only element.</td>
</tr>
<tr>
<td></td>
<td>If an external schema contains an Attributes Group, Design Studio may not recognize it as a valid type.</td>
</tr>
<tr>
<td>Substitution groups</td>
<td>Design Studio ignores this attribute of an element.</td>
</tr>
<tr>
<td>Element Declarations</td>
<td>Design Studio considers any external schema containing element declarations as read-only type definition.</td>
</tr>
<tr>
<td>Unsupported Primitive Types</td>
<td>If an external schema has an element declaration with a primitive type that is not supported, Design Studio considers the type definition as undefined and shown as none.</td>
</tr>
</tbody>
</table>
This chapter provides information about design patterns and guided assistance. It explains how to create your own design patterns and guided assistance in Design Studio and provides information about distributing design patterns and guided assistance.

About the Design Pattern and Guided Assistance SDK Folder

The design pattern and guided assistance SDK folder is included in the Design Studio media pack available on the Oracle software delivery Web site. It includes the following:

- A samples folder
  This folder contains the Pattern folder and a Guided Assistance folder.

  The Pattern folder contains the oracle.communications.sce.pattern.sample.zip archive file, which includes the following projects:
  - oracle.communications.sce.pattern.sample contains the plug-in project (plug-in.xml) that includes a single design pattern.
  - oracle.communications.sce.pattern.sample.feature contains the feature project (feature.xml) that you can use for building and distributing the plug-in project.
  - oracle.communications.sce.pattern.update.site contains an update site project (site.xml) that illustrates how your system administrator can build an update site for delivering your feature to end users.

  The Guided Assistance folder contains the oracle.communications.sce.guidedassistance.sample.zip archive file, which includes the following projects:
  - oracle.communications.sce.guidedassistance.sample contains the plug-in project (plug-in.xml) that includes guided assistance mappings.
  - oracle.communications.sce.guidedassistance.sample.feature contains the feature project (feature.xml) that you can use for building and distributing the plug-in project.
  - oracle.communications.sce.guidedassistance.update.site contains an update site project (site.xml) that illustrates how your system administrator can build an update site for delivering your feature to end users.
See the Design Studio Help for information about importing these projects into your workspace.

- A schema folder

This folder contains the Design Pattern XML schema (DesignPattern.xsd), which is a standard XML Schema document. You can review the contents of this document using any schema or XML editor.

**Working with Design Patterns**

A design pattern is a template containing a set of resources that can be applied to a Design Studio workspace. It can be used to deliver complex sets of preconfigured artifacts that serve some domain-specific function.

**About Design Patterns**

During solution design, designers often create and configure complex sets of related entities and the relationships among them. When these complex sets of tasks are predictable and repeatable, the tasks are often documented in a standard set of guidelines or in a best-practice guide. These sets of tasks are referred to as design patterns because designers can use a template or an established pattern for reproducing the configuration.

In Design Studio, a design pattern is a template containing a set of resources that can be applied to a Design Studio workspace. It can be used to deliver complex sets of preconfigured artifacts that serve some domain-specific function. They enable you to formalize modeling patterns into reusable components that can be applied to various solutions, and eliminate the time and effort required during implementation.

Design patterns enable automation of complex, repeatable tasks, and enable team members with varying levels of skill to complete these tasks. For example, an IT department can develop design patterns to enable network engineers to manage VPNs, such as creating and deleting VPNs, adding sites and removing sites, and so forth.

With Design Studio wizards, you can create and configure new design patterns, and other team members can implement those design patterns. You use these wizards to create complex sets of entities and relationships using a task-oriented user interface. These wizards eliminate the need for coding and complex configuration, which can reduce errors, simplify modeling, and increase productivity. For example, in Design Studio you can apply a design pattern to create an Order Orchestration process, with its related actions and functions. Once created, design patterns can be shared across an organization.

Design patterns are contained in Eclipse plug-in projects and packaged as feature projects. One Eclipse plug-in project can hold any number of design patterns. One Eclipse feature project can contain any number of plug-in projects.

A design pattern includes the following sections:

- **contexts**, which describes where in the Design Studio user interface the design pattern is accessible.

- **manifest**, which describes the resources added to a workspace when the design pattern is applied.

- **projects**, which describes which projects the design pattern can place resources into when the design pattern is applied.
■ **tokens**, which describes the information required from a user to customize the resources in the manifest when the design pattern is applied.

■ **tokenGroups**, which describes the pages in the Design Pattern wizard.

### Creating Design Patterns

The following procedure describes how to create design patterns.

---

**Note:** To create design patterns, the update site used at your organization must include the
oracle.communications.sce.pattern.feature feature, which is included in the updateSite folder in the Design Studio media pack, available from the Oracle software delivery Web site:

https://edelivery.oracle.com

Contact your system administrator if this feature is not available.

---

To create new design patterns:

1. In Design Studio, from the Studio menu, select Design Pattern. The Design Pattern dialog box appears.

2. Expand the Others folder and then expand the Design Pattern Development folder.

3. Select Design Pattern Development and then click Next. The Design Pattern wizard Introduction page appears.

4. Read the information on the Information page, and then click Next. The Select Project page appears.

5. Select a plug-in project to be used for the design pattern development.

6. Click Next. The Plug-in Information page appears.

7. Enter all required information, and then click Next. The Summary page appears.

8. Review the summary information, and then click Finish. Design Studio populates the project with information necessary to build a design pattern. The information includes a manifest, a resource directory, and all plug-in-related configuration for the packaging of the design pattern. Design Studio opens the Design Pattern Development cheat sheet in the Help view.

9. In the Design Pattern Development cheat sheet, click the Click to Begin link and complete the steps in the cheat sheet.

For example, the cheat sheet steps help you with tasks such as copying resources to the project, populating the design pattern manifest, building the design pattern, testing, and distributing the design pattern.

See “Modeling Design Pattern Content”, ”Testing Design Patterns”, ”Securing Design Pattern Information”, and ”Distributing Design Patterns” for more information about each of these steps.
Modeling Design Pattern Content

After you create a design pattern, you can begin to model the design pattern content. Oracle recommends that you use Design Studio to build an example of the design pattern and copy the required resources to the location of your design pattern in your plug-in project.

About Design Pattern Configuration Data
Before configuring design patterns, review the plugin.xml, build.properties, and MANIFEST.MF files. The plugin.xml and build.properties files can be found at the root of the project. The MANIFEST.MF file is in the META-INF directory. These files contain dependencies to the required packages needed for design pattern development. They also include the registration and specific build properties required to build and package your design pattern.

---

**Note:** When you are working on design patterns, many of the artifacts are not visible in standard Design Studio views. Oracle recommends that you add the Package Explorer view to the Design perspective and use that view to manipulate resources. See the Design Studio Help for information.

---

If you are configuring the plug-in project manually, ensure that you configure the following data:

- **MANIFEST.MF**
  
  The name and version of the project and the required plug-ins. The oracle.communications.sce.pattern.core plug-in must be named as a dependency or your design patterns will not be visible when your plug-in is installed.

- **build.properties**
  
  The content that will be visible to other plug-ins after you have packaged your project. If you intend to locate design patterns in other folders, you must manually configure the location in the bin.includes section of the document.

- **plugin.xml**
  
  The design patterns contained by this plug-in. The plugin.xml includes an extension point named oracle.communications.sce.pattern.core.designPattern. This extension point has one entry in the section that points to the location of the pattern. You can add additional entries to package multiple design patterns within a single plug-in project. The Name and ID fields of the registration of a design pattern are not visible to Design Studio users. Rather, they are used only to add clarity to the registration. The ID field must be unique.

About Design Pattern Resources

To add a resource to a design pattern:

- Copy the resource to the location specified for your pattern in the plugin.xml file.
  
  For example, if the design pattern location is specified in the plugin.xml file as:/pattern/myPattern
  
  then you must add resources to this directory or to the subdirectories.

- Add the resource to the pattern.xml document in the manifest section.
All resources must include an **Id** and a **location** value defined in the *pattern.xml* file. The value defined for **Id** must be unique for each resource in the manifest. The value defined for **location** is the relative path of the resource from the location of the design pattern. For example, if the location of the design pattern is:

```
/pattern/myPattern
```

and one of the resources for the pattern from the project root is:

```
/pattern/myPattern/dictionary/pattern.xsd
```

then you define the value for the **location** field for the corresponding resource as:

```
dictionary/pattern.xsd
```

For detailed information about the structure of the manifest and resources in the manifest, see the Design Pattern XML Schema, which is named *DesignPattern.xsd* in the *schema* folder. *DesignPattern.xsd* contains detailed descriptions of the fields in the *pattern.xml* file.

---

**Important**: You must append the extension `.pat` to the Studio Model Entity file names when working with Studio Model Entity resources in a plug-in project. Studio Model Entity files contain Design Studio-specific metadata, such as OSM orders (.order files) and manual tasks (.manualTask files). For example, you would rename the OSM order `myOrder.order` in an original Design Studio project to `myOrder.order.pat` when used in a design pattern plug-in project. You must retain the original name of the Studio Model Entity in the source location of the resource with the .pat extension appended. Design Studio refers to the original entity name while processing the design pattern to update references to entities within the pattern. Failure to append the .pat extension to the resource may result in problem markers and entries in your error log. When working with Data Dictionary companion files, Design Studio automatically deletes the resource from the workspace when it cannot resolve correctly to its associated schema file.

---

**Working with Tokens**

Tokens refer to information collected by the Design Pattern wizard from the user applying the design pattern.

**Token Types**  Token types include:

- Project, which represents the project to which resources are copied when the pattern is applied.

- String, which represents textual input. The text accepted by the token can be constrained by configuring regular expressions.

- Numeric, which represents numeric input.
Note: When defining numeric tokens, you must populate the minimum and maximum values for the token. If you define no values, Design Studio uses 0 as the default and users applying the pattern will not be able to change the value.

- Boolean, which includes true and false values.
- Enumeration, which includes a set of preconfigured choices.
- Entity Reference, which represents the name of a Studio Model Entity in a workspace.
- Element Reference, which represents an element, such as a Data Dictionary element, in a workspace.

For a more detailed description of token types and their configuration, see the Design Pattern XML Schema, which is named DesignPattern.xsd in the schema folder.

Defining Tokens  Tokens can be used in several different ways. They can be:

- Used to change the name or location of a resource by embedding the token in the targetLocation field of the resource.
- Used in the defaultValue field of other tokens to provide intelligent defaults based on previously collected information.
- Embedded in text documents to customize the content of a file.

Regardless of the use, the syntax for the token is always the same. When embedding a token, use the following syntax:

@@<tokenID>@@

Examples: Defining Tokens  Consider that you want to define a token called equipmentName and use it to give an equipment specification in a design pattern a different name. In this example, you would define the targetLocation field for the specification as:

<targetLocation>model/equipment/@@equipmentName@@.equipment</targetLocation>

If a value of opticalDevice is given for the equipmentName token, the targetLocation value in this example expands to the following when the pattern is applied:

model/equipment/opticalDevice.equipment

Tokens can also be defined in other locations of the path in the targetLocation. For example, you might define a token named deviceVendor and use it to expand the previous example:

<targetLocation>model/equipment/@@deviceVendor@@/@equipmentName@@.equipment</targetLocation>

If a value oracle was supplied for the deviceVendor token, the targetLocation value in this example expands to:
Embedding Tokens as Default Values

Tokens can be embedded in the default values of other tokens. For example, you might define a token called `deviceGateway` and define it with the following default value:

```xml
<defaultValue>@@equipmentName@@_gateway</defaultValue>
```

If the value `opticalDevice` were supplied for the `equipmentType` token the default value in this example expands to:

```xml
opticalDevice_gateway
```

When using embedded tokens as default values for other tokens, ensure that the embedded token appears in an earlier token group than where it is used. If a value has not been assigned before the token is displayed to a user, the embedded token will be displayed in the `defaultValue` field.

Restricting Token Values

When using a String token, restrict the valid input using regular expressions. Some character-based input is not valid for use in a token definition. For example, if you use a token as a file name, then it must contain only characters that are valid for a file name. If you use a token as a Studio Model Entity name or in an XML document, you must restrict the use of XML special characters. Oracle recommends that you employ a restrictive policy when defining regular expressions to avoid invalid file names and corrupted XML files. Embedded tokens are the most common place where errors are introduced into a design pattern. If a design pattern is not working properly, first ensure that all tokens are properly replaced when the pattern was applied.

About Element References

Element reference tokens return XML instead of a simple value. Therefore, you cannot embed element references in default values or in target locations. Design Studio returns XML to ensure that the token can be embedded in a Studio Model Entity document, where it replaces a reference to an element.

The format of the return XML is:

```xml
<com:entity>--from selection--</com:entity>
<com:entityType>--from selection--</com:entityType>
<com:relationship>--from token configuration--</com:relationship>
<com:element>--from selection--</com:element>
<com:elementType>--from token configuration--</com:elementType>
```

Use the token to replace a section of XML in a Studio Model Entity document with the corresponding structure.

Working with Cheat Sheets

Design Studio supports cheat sheets, which refers to the integration of documented procedures with wizards in the application. Cheat sheets are XML documents that can be interpreted by the Eclipse Cheat Sheet framework, and developers can map cheat sheets to specific points in the Design Studio user interface (for example, in editors and views). You access the cheat sheets that are relevant to current tasks, and complete those tasks using the included instructions. Cheat sheets enable you to find documentation for relevant solution design procedures and facilitate the learning of those procedures.
For example, you can use cheat sheets with design patterns to describe the resources added to a workspace and to assist users with any manual steps required after a design pattern is applied. Cheat sheets are not mandatory for design patterns, but they are recommended.

You can develop and edit cheat sheets using the Eclipse Cheat Sheet editor.

For information about creating and developing cheat sheets, see "Building Cheat Sheets in Eclipse" on the Oracle Technology Network:

http://www.oracle.com/technetwork/articles/entarch/eclipse-cheat-sheets-092351.html

**Testing Design Patterns**

You must test design patterns after you build them. You can test design patterns by running them in the Design Pattern wizard.

To test design patterns:

1. In Design Studio, from the Studio menu, select Design Pattern.
   The Design Pattern wizard appears.
2. Select the Select a Design Pattern from File option.
3. Click Browse and locate and select the design pattern that you want to test.
4. Click Next.
   If the selected design pattern contains structural problems (such as schema validation errors or invalid XML), an error message appears. The error message displays a specific schema validation error and the line and column in the document where the error exists. You must fix all errors before testing a design pattern.
5. Navigate through the wizard to ensure that the design pattern works as intended.
6. When the wizard completes, verify that all token replacements work as intended.
   For example, you may need to correct any misspelled tokens in the pattern.xml file and all other documents.

**Securing Design Pattern Information**

Design patterns have no security for the information they include, and any user can apply design patterns. Do not include sensitive information in design patterns. If sensitive information is needed to complete the configuration of a design pattern, include a cheat sheet to run after the pattern is applied to manually configure the sensitive information.

Oracle recommends that you use model variables in design pattern templates to replace any sensitive information. Design Studio users can be prompted to configure a sensitive model variable with the name used in the template. See the Design Studio Help for information about working with model variables.

**Distributing Design Patterns**

Design patterns are developed in plug-in projects. Plug-in projects are grouped into features, and features are made available to users through update sites.

To distribute design patterns, you create a feature project using the Feature Project Creation wizard and add the plug-in project to the feature project. When you are
finished, contact your system administrator to request that the new feature be added to the Design Studio update site.

See *Eclipse Plug-in Development Environment Guide* for information about using the Feature Project Creation wizards. The sample included in the Design Studio media pack demonstrates how to configure a feature project.

## Applying Design Patterns

Design Studio users can apply design patterns by accessing the Design Pattern wizard in the Design Studio user interface.

---

**Note:** Design patterns may overwrite resources or skip existing resources. Re-running a design pattern with the same input may result in a different output depending on the current state of the workspace and the configuration details of the design pattern.

Some design patterns can overwrite existing resources. Oracle recommends that you use a version control system and employ a backup strategy to ensure against data loss.

---

To apply a design pattern:

1. In Design Studio, from the **Studio** menu, select **Design Pattern**.
   
   The Design Pattern wizard appears.

2. Select a design pattern to apply, and then click **Next**.

3. Enter information into the wizard, as prompted.
   
   The Design Pattern wizard prompts you for all information (the tokens) required to apply the pattern.

4. Navigate through the wizard to completion.
   
   After the wizard collects all of the information, Design Studio copies the resources from the manifest into your workspace. A resources folder is given a target location in the manifest and is copied to that location using the target name derived from the data collected in the wizard.

   Design Studio updates the references between the Studio Model Entities copied into the workspace, based on the names defined in the manifest.

   Design Studio then locates any tokens embedded in text documents in the manifest (Studio Model Entities, XML, XSLT, XQuery, Java, and so forth) and replaces them with the values you specified in Design Pattern wizard.

5. (Optional) In the Help view, use the cheat sheet for information and guidance related to any required additional steps.
   
   Design Studio might start an Eclipse cheat sheet after the design pattern is applied, if a cheat sheet is included in the pattern.

After the design pattern is applied, it is indistinguishable from other configuration in the Design Studio workspace. You can modify the configuration to meet any specific requirements.
Working with Guided Assistance

Design Studio guided assistance is a range of context-sensitive learning aides mapped to specific editors and views in the user interface. For example, when working in editors, you can open the Guided Assistance dialog box for Help topics, cheat sheets, and recorded presentations that are applicable to that editor.

When working with guided assistance, you can review the learning aids delivered with Design Studio, and you can create your own and map them to projects and entities by using design patterns or by defining values for attributes directly in the guided assistance extension point.

About the Guided Assistance Dialog Box

You can access learning aids delivered in Design Studio by opening the Guided Assistance dialog box, which is available from the Studio menu, the main tool bar, and from the Cartridge view context menu.

The learning aids included in the Guided Assistance dialog box are organized into categories that reflect a specific domain. For example, the Order and Service Management Project directory includes a category called Order, which includes learning aids that help you define and configure orders.

The Guided Assistance dialog box is organized in the following hierarchy:

- The root level contains global guided assistance that is not specific to any cartridge project.
- The second level is organized by project type and contains guided assistance for specific cartridge projects.
- The third level contains entity-specific guided assistance for each project type.
- The fourth level contains folders that include learning aids that are specific to functionality within an entity.

Folders appear only when there is guided assistance available at the corresponding folder level. When you first open the Guided Assistance dialog box, the hierarchy expands to the folder relevant to the context in focus. If no guided assistance is mapped to the active context, the hierarchy will appear fully collapsed. You can navigate to any folder level in the hierarchy, regardless of the context in focus.

Working with Guided Assistance Design Patterns

You can create and implement new guided assistance by using design patterns delivered with Design Studio.

These design patterns are preconfigured for guided assistance development and can help you create a directory structure where Design Studio users can save guided assistance learning aids. After applying a guided assistance design pattern, this directory structure appears in the plugin.xml file.

Creating Guided Assistance Using Design Patterns

The following procedure describes how to create your own guided assistance using design patterns.
To create new guided assistance using design patterns:

1. In Design Studio, from the Studio menu, select Design Pattern.
   The Design Pattern dialog box appears.

2. Expand the Others folder.
   A list of guided assistance folders appears, one for each Design Studio project type.

3. Expand the folder for the project type for which you want to add guided assistance.
   For example, if you are adding guided assistance for model projects, expand the Guided Assistance Design Pattern for Model folder.

4. Select the design pattern and click Next.
   The Design Pattern wizard Introduction page appears.

5. Read the information on the Information page, and then click Next.
   The Select Project page appears.

6. Do one of the following:
   - To use an existing project for guided assistance development, select a plug-in project in the Guided Assistance Plug-in Project field.
   - To create a new project for guided assistance development, click New.

7. Click Next.
   The Plug-in Information page appears.

8. Enter all required information, and then click Next.
   The Summary page appears.

9. Review the summary information, and then click Finish.
   Design Studio populates the project with information necessary to build the guided assistance for the designated project type. The information includes a manifest and a guidedAssistance directory which contains the folder structure for the project type and all plug-in related configuration for the packaging of the guided assistance. When you finish the wizard, Design Studio opens the Guided Assistance cheat sheet in the Help view.

10. (Optional) Modify the cheat sheets in the oracle.communications.sce.guided.assistance.feature.
    The cheat sheets should provide help to users who apply the guided assistance design pattern. The information should describe how to copy guided assistance learning aids to the folder structure created by applying the design pattern.

---

**Note:** To create guided assistance, the update site used at your organization must include the oracle.communications.sce.guided.assistance.feature feature, which is included in the updateSite folder in the Design Studio media pack, available from the Oracle software delivery Web site:

https://edelivery.oracle.com

Contact your system administrator if this feature is not available.
Working with Guided Assistance

You use the guided assistance extension point (com.mslv.studio.core.ui.studioGuidedAssistance) to register the learning aid content locations.

In the plugin.xml file, you define attributes for the extension point to register guided assistance (for example, cheat sheets, HTML files, and Help documents) content locations that are applicable to:

- All project types (globalGuidedAssistance)
- A specific project type (cartridgeGuidedAssistance)
- A specific entity type (entityGuidedAssistance)
- A specific functionality in an entity type (guidedAssistanceContent)

### Table 2–1 Guided Assistance Extension Point Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Element Used In</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>guideName</td>
<td>globalGuidedAssistance</td>
<td>Enter the display name that appears in the Guided Assistance dialog box for the learning aid. If you define no value for helpContextId, the name you define here is displayed for the contentLocation, but only when the location refers to a single file; otherwise this value is ignored.</td>
</tr>
<tr>
<td></td>
<td>cartridgeGuidedAssistance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>guidedAssistanceContent</td>
<td></td>
</tr>
<tr>
<td>contentLocation</td>
<td>globalGuidedAssistance</td>
<td>Enter the folder location for the learning aid.</td>
</tr>
<tr>
<td></td>
<td>cartridgeGuidedAssistance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>guidedAssistanceContent</td>
<td></td>
</tr>
<tr>
<td>helpContextId</td>
<td>globalGuidedAssistance</td>
<td>Enter a unique ID that represents the learning aid. Design Studio uses this value to display the appropriate Help page. Optionally, you can specify the Help URL. Use the following format to define a Help context ID: pluginId.contextId For example: com.company.product.help.myContextId</td>
</tr>
<tr>
<td></td>
<td>cartridgeGuidedAssistance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>guidedAssistanceContent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>entityGuidedAssistance</td>
<td>Enter a fully qualified ID of the project type to which the learning aid is related. The projectType extension can be defined in any plug-in (that is, it does not need to be defined in the same plug-in as cartridgeGuidedAssistance).</td>
</tr>
<tr>
<td>projectTypeId</td>
<td>cartridgeGuidedAssistance</td>
<td>(optional)</td>
</tr>
<tr>
<td></td>
<td>entityGuidedAssistance</td>
<td></td>
</tr>
<tr>
<td>entityId</td>
<td>entityGuidedAssistance</td>
<td>Enter the fully qualified name of a modelType extension. The extension can be defined in any plug-in (that is, it does not need to be defined in the same plug-in as entityGuidedAssistance).</td>
</tr>
<tr>
<td>folderId</td>
<td>guidedAssistanceContent</td>
<td>Enter the name of the folder in which the learning aid should appear in the Guided Assistance dialog box.</td>
</tr>
</tbody>
</table>
Guided Assistance Extension Point Example

Example 2–1 shows how you can configure the attributes for the guided assistance extension point:

Example 2–1 Example: Guided Assistance Extension Point

```xml
<extension
    point="com.mslv.studio.core.ui.studioGuidedAssistance">
    <globalGuidedAssistance
        contentLocation="guidedassistances/"
        guideName="Global Guides Location">
    </globalGuidedAssistance>

    <cartridgeGuidedAssistance
        contentLocation="guidedassistances/ModelProject/"
        guideName="Data Dictionary Wizard Help"
        helpContextId="com.mslv.studio.core.help.DataDictionaryWizard"
        projectTypeId="com.mslv.studio.core.datadictionary.project">
    </cartridgeGuidedAssistance>

    <entityGuidedAssistance
        entityId="com.mslv.studio.core.dataDictionary"
        projectTypeId="com.mslv.studio.core.datadictionary.project">
    <guidedAssistanceContent
        contentLocation="guidedassistances/ModelProject/DataSchema/"
        guideName="Data Schema Guides">
    </guidedAssistanceContent>
    </entityGuidedAssistance>

    <cartridgeGuidedAssistance
        contentLocation="guidedassistances/OSM/"
        guideName="OSM Guides"
        projectTypeId="com.mslv.studio.provisioning.project">
    </cartridgeGuidedAssistance>

    <entityGuidedAssistance
        entityId="com.mslv.studio.provisioning.process"
        projectTypeId="com.mslv.studio.provisioning.project">
    <guidedAssistanceContent
        contentLocation="guidedassistances/OSM/order/"
        folderId="com.mslv.studio.provisioning.order">
    </guidedAssistanceContent>
    <guidedAssistanceContent
        contentLocation="guidedassistances/OSM/process/creationTask/"
        folderId="orderCreationTask">
    </guidedAssistanceContent>
    </entityGuidedAssistance>

    <entityGuidedAssistance
        entityId="com.mslv.studio.provisioning.order">
    <guidedAssistanceContent
        contentLocation="guidedassistances/OSM/order/"
    </guidedAssistanceContent>
    <guidedAssistanceContent
        contentLocation="guidedassistance/OSM/video/722_whats_new.htm"
        guideName="Design Studio 7.2.2 What’s New Video">
    </guidedAssistanceContent>
    </entityGuidedAssistance>
</extension>
```
Distributing Guided Assistance

You save guided assistance in Eclipse plug-in projects. Plug-in projects are grouped into features, and your system administrator can make these features available to other users by adding the feature to your Design Studio update site.

To distribute guided assistance:

1. From the File menu, select New and then select Project.
   The New Project wizard appears.
2. Expand the Plug-in Development folder.
3. Select Feature Project, and then click Next.
   The New Feature wizard appears.
4. Enter the information required by the wizard, and then click Finish.
   The new feature appears in the Feature editor.
5. Click the Plug-ins tab.
6. Click Add.
7. Add the plug-in project in which you saved the guided assistance.
   You can add any number of projects to the feature.
8. Contact your system administrator to request that the new feature be added to the Design Studio update site.

See Eclipse Plug-in Development Environment Guide for information about using feature projects. The samples included in the Design Studio media pack demonstrate how your system administrator can configure feature projects.
Working with Source Control

This chapter provides information to enable you to collaborate in teams by using a source control system to share projects and describes which files must be source controlled in Oracle Communications Design Studio.

About Source Control

Oracle recommends that you use a source control system to manage the quality of service fulfillment design solutions. With a source control system, you can share projects across development teams, submit changes to various projects, and import projects into Design Studio. Oracle recommends that you always use a source control system in your workflow, even if only one team member is working on a development project. Using source control, you can track changes to systematically correct mistakes, roll back to previous versions, and support quality control. Design Studio can integrate with source control software, such as Apache Subversion, GIT, and Concurrent Versions System (CVS).

For team development projects, you install the source control software on a server. A server installation enables multiple developers to remotely access the hosted repository. For development projects with one developer, you can install the source control software locally on the individual computer. A local installation enables a single user to access the repository on the computer’s file system. Commands execute directly without the need for a server.

About Source Control Strategies for Design Studio Files

Oracle recommends that developers work in a source control system when developing cartridges in Design Studio. Oracle Enterprise for Eclipse provides support for integrating with source control systems (plug-ins are available for most common source control systems). The behavior of Design Studio when used in an environment where the files are backed by a source control system depends on the source control system and the source control team plug-in being used. For more information about working in the team environment, see Eclipse Workbench User Guide. See the Eclipse Marketplace page for more information about supported source control solutions:


Table 3–1 describes the structure of the directories and the files in a Design Studio and recommends a source control management strategy.
### Table 3–1 Design Studio Source Control Management

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Description</th>
<th>Source Control Management</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ProjectDir/</code></td>
<td>Project's top level directory.</td>
<td>Source control all files directly under this directory.</td>
</tr>
<tr>
<td><code>ProjectDir/cartridgeBin/</code></td>
<td>Contains deployable archive files</td>
<td>Source control the directory but do not source control the contents.</td>
</tr>
<tr>
<td><code>ProjectDir/doc/</code></td>
<td>Contains documentation files.</td>
<td>Source control.</td>
</tr>
<tr>
<td><code>ProjectDir/generated/</code></td>
<td>Contains generated artifacts of the build process.</td>
<td>Source control. However, do not source control the <code>src</code> folder or its contents.</td>
</tr>
<tr>
<td><code>ProjectDir/generated/src/</code></td>
<td>Contains generated artifacts of the build process.</td>
<td>Source control the directory but not its contents.</td>
</tr>
<tr>
<td><code>ProjectDir/integrityLib/</code></td>
<td>Contains Design Studio for Network Integrity JAR files that are included in the Network Integrity server Enterprise Archive (EAR). These JARS are in the project's classpath.</td>
<td>Source control the directory, but do not source control the files in this directory.</td>
</tr>
<tr>
<td><code>ProjectDir/integrityLib/packaged</code></td>
<td>Contains JAR files that are created by Design Studio for Network Integrity and which are packaged into the cartridge IAR file. The JAR files are added to the Network Integrity EAR when the cartridge is deployed. These JAR files are in the project's classpath.</td>
<td>Source control the directory, but do not source control the files in this directory.</td>
</tr>
<tr>
<td><code>ProjectDir/lib/</code></td>
<td>Contains JAR files.</td>
<td>Source control. The <code>mds.mar</code> file is output to this directory. Do not source control the <code>mds.mar</code> file.</td>
</tr>
<tr>
<td><code>ProjectDir/mdsArtifacts/</code></td>
<td>Contains files that contribute to the UI Hints infrastructure.</td>
<td>Source control the directory and the following files:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>MDSAvailablePagePanels.xml</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>MDSAvailablePagePanels.xsd</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>MDsMetaData.xml</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not source control the the remaining files in this directory.</td>
</tr>
<tr>
<td><code>ProjectDir/model/</code></td>
<td>Contains files that are used to persist the information about Cartridges, Actions, Processors, Model Collections, and Address Handlers.</td>
<td>Source control.</td>
</tr>
<tr>
<td><code>ProjectDir/out</code></td>
<td>Contains output classes.</td>
<td>Do not source control.</td>
</tr>
</tbody>
</table>
Table 3–1  (Cont.) *Design Studio Source Control Management*

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Description</th>
<th>Source Control Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectDir/src/</td>
<td>Contains the user-supplied code for the cartridge.</td>
<td>Source control.</td>
</tr>
</tbody>
</table>
Deploying Cartridges to Environments

This chapter provides information about the Cartridge Management Tools (CMT) utility and about other tools that you can use when deploying cartridges to run-time environments.

Deploying Cartridges to Run-Time Environments with the Cartridge Management Tools Utility

The CMT enables you to deploy cartridges to run-time environments outside of the Oracle Communications Design Studio environment. The CMT is bundled with the Design Studio Media Pack on the Oracle software delivery Web site.

**Note:** Oracle recommends that you deploy to production environments using a controlled and scripted process. This process should be verified in a staging environment prior to execution against a production environment. Though Design Studio enables you to deploy cartridges to design and test environments consistently across all Oracle Communications features, Oracle recommends that using the utility to deploy to production run-time environments.

To deploy cartridges to run-time environments using CMT:

1. Navigate to Oracle software delivery Web site:
   
   https://edelivery.oracle.com

2. Download and extract the Design Studio Media Pack.

3. Open the `cartridge_management_tools` folder.

4. Extract `cartridge_management_tools.jar`.

5. Open the `build.properties` file and update the file with environment-specific variables.
   
   For example, update Web service connection information for deploying and undeploying cartridges, model variable values to be used at deploy time, and so forth.

6. Add the files in the `lib` folder (in the `cartridge_management_tools` file) to the classpath.
   
   Adding libraries to the classpath can prevent exceptions from being thrown during Ant script execution.
7. From a command prompt, navigate to the directory where the cartridge_management_tools.jar file is extracted and run the following Ant commands, as applicable:
   - `ant`: displays a list of available targets.
   - `ant -lib \lib -f build.xml deploy-cartridge`: Deploys a cartridge to a run-time environment.
   - `ant -lib \lib -f build.xml undeploy-cartridge`: Undeploys a cartridge from a run-time environment.

**Working with Additional Cartridge Deployment Tools**

Oracle recommends using the CMT for automated, scripted, and command line cartridge deployment for all the applications supported by Design Studio. Under specific circumstances, applications might recommend other tools for cartridge deployment, such as the following:

- **Service Activation Deployment Tool (SADT)**, used to deploy SAR files to ASAP run-time environments. See *ASAP Cartridge Development Guide* for more information.
- **Cartridge Deployer tool**, used to deploy JAR files to UIM run-time environments. See *UIM System Administrator’s Guide* for more information.
- **XML Import/Export tool**, which is used to manage data in the OSM database. See *OSM System Administrator’s Guide* for more information.

See the Oracle Communications application documentation for specific instructions and applicability notes and to determine the level of support for application-specific tools.
This chapter provides an overview of Oracle Enterprise Packet for Eclipse and Java Development Tools.

**Working with Oracle Enterprise Packet for Eclipse**

Oracle Communications Design Studio supports Oracle Enterprise Pack for Eclipse, a set of certified plug-ins designed to help you develop and debug Java EE applications that can be deployed on Oracle WebLogic Server from the Eclipse.

Design Studio uses the Eclipse platform as a product framework and as an integrated development environment (IDE) to support plug-in architecture and customizations. Eclipse provides a GUI to manage and configure data across Oracle Communications products.

Eclipse supports application development tool construction, independent tool vendors, GUI and non-GUI application development, numerous content types (including Java, HTML, C, and XML), tool integration, and use of Java language for writing the tools.

For more information about installing Oracle Enterprise for Eclipse, see *Design Studio Installation Guide*.

---

**Note:** To run Oracle Enterprise for Eclipse, system administrators must install the correct Java Runtime Environment and Java Developer Kit. See *Design Studio Installation Guide* for more information.

Oracle Enterprise for Eclipse includes a number of tools that are useful for middleware development and includes all of the required features to support Design Studio. These tools complement Design Studio features to provide a more complete design environment for building solutions.

**About Java Development Tools**

Java Development Tools (JDT) provide a set of workbench plug-ins that add the capabilities of a full-featured Java IDE to the Eclipse platform. JDT plug-ins provide APIs that can be further extended by other tool builders. Additionally, the JDT includes a built-in Java compiler that compiles Java code and creates error messages when compilation fails.
About Database Development Tools

Oracle Enterprise for Eclipse provides tools to help you develop applications that use Oracle Database. These tools include:

■ Support for the integration of Oracle Database with Eclipse Data Tools Platform.
■ Diagram viewers for visualizing database schemas and object-relational mappings.
■ Database Explorer, which supports data editing, data load/extract, and Data Definition Language generation.
■ SQL tools that support SQL editing and execution and stored procedures.

About Application and WebLogic Server Tools

Oracle Enterprise for Eclipse editors and optimized development tools for application server development simplify work with products like WebLogic Server by providing:

■ Fast, iterative deployment for local and remote servers.
■ Support for JAX-WS Web services (editor and configuration support).
■ Oracle Enterprise for Eclipse visual deployment descriptor editors for *-jms.xml, weblogic.xml, weblogic-application.xml, weblogic-ejb.jar.xml, faces-config.xml, and persistence.xml files, and a JSR 88 deployment plan editor.

About Web Application Tools

Oracle Enterprise for Eclipse Web application tools simplify working with technologies like JSF, JSP, CSS, ADF, and others.

Web Applications

The following Web application tools simplify analyzing and visualizing dependencies to reduce run-time debugging and to improve code quality:

■ AppXRay and AppXaminer offer compiler-level awareness of Java, Oracle Application Development Framework (Oracle ADF), HTML, CSS, JSP, JSTL, and JSF at design time, with capabilities in code and annotation completion, code navigation, dependency visualization, consistency checking with generated classes and configuration files, pre-build error checking, and validation.
■ An enhanced Eclipse Web Tools Platform (WTP) Web Page Editor (WPE) includes a Smart Property Sheet to simplify tag configuration and data binding. WPE also includes localization support, JSF, JSP, JSTL, and CSS/HTML, and a Tag and Data Palette.

Web Tools Platform

The WTP extends the Eclipse platform to simplify Web and Java EE application development. WPT provides the following tools to simplify deploying, running, and testing applications:

■ Core Web Tools Platform (EJB Tools, Java EE Tools, Server Tools)
■ JavaServer Faces Tools
■ Data Tools Platform Project
Oracle ADF Tools
Oracle Enterprise for Eclipse provides design-time support for application development with Oracle ADF. You can create applications that leverage Oracle ADF Faces and Task Flows, validate and refactor Oracle ADF dependencies using AppXRay, deploy and debug with Oracle WebLogic Server, and create Oracle ADF Libraries for application reuse.

The following Oracle ADF tools simplify Java EE development by minimizing the need to write code that implements the application’s infrastructure:

- Oracle ADF Server Extensions to configure WebLogic Server and Eclipse for ADF development
- ADF Project Templates and Facets
- ADF-enabled JSP Templates
- ADF design time support
  - WPE enhancements for ADF source development
  - Smart Property Sheet to simplify tag configuration and data binding
  - Tag Palette enabled for ADF Faces and Data Visualization Tools (DVT) components
  - Tag editors for drag and drop configuration of ADF tags
- AppXRay support for ADF tags

About JPA and Oracle Coherence Tools
The following Oracle Enterprise for Eclipse tools help you create applications that map objects to relational databases.

Java Persistence API (JPA) Tools
JPA tools simplify working with the Java programming language framework to manage relational data in applications using Java Platform, Standard Edition, and Enterprise Edition. These tools enable you to:

- Use object relational mapping tools for Java Persistence API
- Generate entities from schema
  - Start with any database connection
  - Create entities based on table relationships
  - Define new entity associations
- Generate entities from POJO
  - Annotate existing Java class
  - Map POJO fields and properties to database schema
- Use the Entities editor, which includes:
  - A design view to display and edit existing entity relationships
  - Hyperlinked navigation to entity source code
- Use the JPA Details view, enabling you to:
  - Edit entity properties and relationships from the Entities editor
- Support EclipseLink/TopLink, Kodo, OpenJPA, and generic JPA providers
**Oracle Coherence Tools**

Oracle Enterprise for Eclipse supports Oracle Coherence. You can run, deploy, and debug Coherence servers from Eclipse, as well as create and configure projects, and leverage visual editors for cache configuration and override descriptors.

**About Third-Party Tools**

Design Studio and the Eclipse platform support third-party tools that are helpful for readiness, testing, and other administrative activities. For example, you might use:

- soapUI, as a test tool for calling Web services.
- HermesJMS, as a test tool for sending and receiving JMS messages.
- SQL Developer, as a utility for maintaining the database.
- XQDT for Eclipse or oXygen XML Developer, as an XQuery editor.