Oracle® Communications
Unified Inventory Management
Migration Guide
Release 7.2
E28809-02

June 2012
# Contents

Preface ................................................................................................................................. vii  
   Audience ........................................................................................................................ vii  
   Documentation Accessibility ......................................................................................... vii  
   Related Documentation ............................................................................................... vii  

1 Overview  
   Static Extensibility ....................................................................................................... 1-2  

2 Deployment Architecture  
   Deployment Architecture Changes ............................................................................ 2-1  

3 Design Studio Tasks  
   About Cartridge Migration ......................................................................................... 3-1  
   Characteristic Renaming Rules .................................................................................. 3-1  
   Cartridge Migration Process Overview ...................................................................... 3-2  
      Cancellation During Migration ................................................................................ 3-3  
   Manual Reset of Layouts ............................................................................................. 3-3  
   Changing JDOQL to JPQL on Query Characteristics .................................................. 3-4  
   Migrating Custom Rule Sets ....................................................................................... 3-4  
      Packaging Changes ................................................................................................ 3-4  
      Import Statement .................................................................................................... 3-4  
      Use InventoryConfigurationVersion as a Parameter in Rulesets ................. 3-4  
      Connect if the Entity is Stale .................................................................................. 3-5  
   Cartridge Deployment .................................................................................................. 3-6  

4 Database Migration  
   Database Upgrade Reports ............................................................................................ 4-1  
   SEQUENCE Table Migration ....................................................................................... 4-1  
      Ad Hoc Sequence Generation ................................................................................ 4-2  
      Specification-based Sequence .............................................................................. 4-2  
      Context-based Sequence ....................................................................................... 4-3  
   Automatic ID Generation ............................................................................................ 4-4  
   Custom ID Generation ................................................................................................. 4-4  
   Default ID Generation ................................................................................................. 4-4
5 Migrating Custom Java Code

- Class Names and Package Changes ................................................................. 5-1
- Returned Collections ...................................................................................... 5-2
- Finder ........................................................................................................... 5-2
  - Finder.findByEntity .................................................................................. 5-3
  - Finder.setRange ....................................................................................... 5-3
- InventoryFinder ............................................................................................. 5-3
- Changing JDOQL to JPQL ............................................................................ 5-4
- Eager Fetch Changes .................................................................................... 5-4
- Consumer Model Changes ............................................................................ 5-5
- Entity Life Cycle Listener ............................................................................. 5-5
  - Disabling /Enabling life cycle listeners ....................................................... 5-6
  - Entity-specific configuration ...................................................................... 5-6

6 Web Services

- Framework Changes ..................................................................................... 6-1
- Changes to the Web Services Framework Schema ........................................ 6-3
- Development Environment Changes ............................................................ 6-6
  - Recommended Eclipse .classpath file .......................................................... 6-6
  - Build Change .............................................................................................. 6-7
  - .java File Changes ..................................................................................... 6-7

7 Migrating Security Data

- Migrating Users, Partitions and Roles ........................................................... 7-1
  - Migrating Users ......................................................................................... 7-1
  - Migrating Partitions ................................................................................... 7-2
    - Creating WebLogic Server Groups .......................................................... 7-2
    - Assigning Users to Groups ..................................................................... 7-2
  - Migrating Roles ........................................................................................... 7-2
- Migrating Security Policies ............................................................................ 7-3
  - Migrating Authorization Policies for Work Areas ....................................... 7-3
  - Migrating Authorization Policy for UI Actions .......................................... 7-3
- Entity-level Authorization ............................................................................. 7-4

8 User Interface Changes

- Localization .................................................................................................. 8-1
- Customization ............................................................................................... 8-1
- Organizer ...................................................................................................... 8-1
- Externally Referenced URL .......................................................................... 8-1
- Bread Crumbs ............................................................................................... 8-1
- My Navbar ................................................................................................. 8-2
- My Desktop ................................................................................................. 8-2
- Find Links ................................................................................................. 8-2
9 Data Export Changes

Entity Class No Longer Used .................................................................................................................. 9-1
Boolean Expression No Longer Used ...................................................................................................... 9-1
Use Queries Instead of Classes and Boolean Expressions........................................................................ 9-1
RelationshipToInclude Parameter ........................................................................................................ 9-1
UIM Meta Entities .................................................................................................................................... 9-2
Sample Export Control Files .................................................................................................................. 9-2
Preface

This guide provides information about migration tasks associated with upgrading from a previous version of Oracle Communications Unified Inventory Management.

Audience

This guide is intended for users who are responsible for planning or executing upgrades from previous releases to UIM 7.2.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documentation

For more information, see the following documents in the Oracle Communications Unified Inventory Management documentation set:

- **UIM Concepts**: Provides an overview of important concepts and an introduction to using both UIM and Design Studio.

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

- **UIM System Administrator’s Guide**: Describes maintenance and administrative tasks such as managing user security.

- **UIM Developer’s Guide**: Explains how to customize and extend many aspects of UIM, including the schema, user interface, rule sets, Web services, life-cycle management, topology, security, and localization.

- **Oracle Communications Information Model Reference and Unified Inventory Management Information Model Reference**: Describes the Oracle Communications Information model and the UIM extensions to it. These documents include
information about entities and data elements. It also explains patterns that are common across all entities.

■ **UIM Base Cartridges Guide**: Provides descriptions of entities included in base cartridges. These entities are used for all technology domains. Examples include telephone numbers, units of measure, and base rule sets.

■ **UIM Technology Pack Implementation Guides**: Describe the content of product technology packs as well as configuration guidelines and implementation considerations.

For step-by-step instructions for performing tasks, log into each application to see the following:


■ **UIM Help**: Provides step-by-step instructions for tasks you perform in UIM.
Oracle Communications Unified Inventory Management (UIM) 7.2 has been upgraded to conform more closely to other Oracle applications based on the Fusion platform. Older technologies used in previous releases have been. For example, the persistence architecture has been changed from Kodo to EclipseLink, the older web framework Tapestry has been replaced by ADF, and Weblogic version has been upgraded to FMW 11g.

This document outlines the steps required to migrate successfully to UIM 7.2 and provides links to other documentations that supply additional information. The document also the deployment architecture for this release and points out the differences from the previous release.

Figure 1–1 depicts the tasks that may need to be completed during the upgrade. Depending on how UIM is deployed and used, you may not need to complete all of the tasks.
You can extend the UIM data model by adding to the metadata. For example, you can add new entities to the model, add new attributes to the UIM entities, add new enumerations, or extend the life cycle transitions of entities.

Extensions to the data model are not backward compatibility with previous releases. You must perform these extensions again for every new UIM release.

The process of extending the data model has changed for UIM 7.2. See *UIM Developer’s Guide* for details.
A number of changes have been made to the deployment architecture of Oracle Communications Unified Inventory Management (UIM) 7.2. This chapter outlines some of the most important changes.

**Deployment Architecture Changes**

*Figure 2–1* illustrates the deployment architecture of UIM 7.1, the previous release.

*Figure 2–2* illustrates the deployment architecture of UIM 7.2.
The following list describes the major differences between the deployment architectures in the previous release and UIM 7.2.

- In UIM 7.1, there was only one EAR file, `inventory.ear`. In UIM 7.2, `inventory.ear` is still delivered, but, the following EAR files are also delivered.
  - `inventory-adapter.ear`. Application used for deploying cartridges from Design Studio or the Cartridge Deployment tool.
  - `uim_custom_lib.ear`. Contains custom code that is deployed in cartridges. In previous releases, this code was added to `inventory.ear`.
  - `custom.ear`. This is an empty application that users can optionally use as a starting point to add custom Web services. The `custom-ws.war` file is now packaged in `custom.ear` instead of `inventory.ear`.

- In previous releases, all third-party jars were included in the `APP-INF/lib` of `inventory.ear`. Now, third-party jars have been modularized as a shared library called `uim_external_lib.jar`. They are no longer packaged in `inventory.ear`.

- The `inventory.ear` file no longer contains `baseData.war`. All the necessary base data can now be loaded by deploying the base cartridges.

- The `inventory.ear` no longer contains the `test.war`. It was decided that the test clients were internal and there is no need to deliver them.

- In 7.1, `inventory.ear`, `APP-INF/lib` contained the following jars:
  - `mslv-core.jar`. This jar contained the classes corresponding to all the managers and their implementations. This file is now renamed to `uim-managers.jar` and resides in `uim_core_lib.ear`, which is installed as a shared library.
  - `mslv-api-framework.jar`. This jar contained the classes corresponding to the UIM framework. This file is now renamed to `uim-api-framework.jar` and resides in `uim_core_lib`, which is installed as a shared library.
- **inventory.ear**. This jar contains classes corresponding to all the entities and their implementation. This file has been renamed to **inventory.ear** and still resides in the **APP-INF/lib** of all the EAR files.

  - The complete list of jars in **uim_core_lib.ear** are
    - uim-webservices-framework.jar
    - uim-webservices-framework-xsd.jar
    - uim-webservices-base.jar
    - uim-tools.jar
    - uim-managers.jar
    - uim-entity-xmlbean.jar
    - uim-caps.jar
    - uim-api-framework.jar
    - platform_managers.jar
    - groupenabled_caps.jar
    - core_caps.jar
    - consumable_caps.jar
    - characteristic_caps.jar
    - capacity_caps.jar

  - The **inventory.ear** in 7.2 now contains a new WAR file called **UIMServiceFullfillment.war**. This file has the new Web services created in this release.

  - In release 7.2, all the components provided by the Platform team are deployed as shared libraries. These shared libraries have frameworks for various modules and can be shared by different applications.
You must complete several tasks in Oracle Communications Design Studio to migrate content for Oracle Communications Unified Inventory Management (UIM) 7.2. In particular, you must migrate the content of all cartridges that will be deployed to UIM 7.2.

About Cartridge Migration

In this release, the Design Studio for UIM plug-in has adopted the common data dictionary as a repository and tool for defining characteristics and default data elements. In previous releases, characteristics were defined separately in characteristic specifications. In the current release, characteristics are data elements that are flagged as characteristics. Entity specifications now reference these data elements from the data dictionary.

Because of this change and other infrastructure changes, it is necessary to migrate cartridges to the 7.2.0 release. Existing inventory cartridge data is incompatible with the new data formats. A cartridge from a previous release cannot be deployed into a 7.2 environment.

**Caution:** Failing to migrate a cartridge will result in data corruption. Read this entire section before beginning the migration process.

The Design Studio Cartridge Migration feature enables you to migrate older cartridges to 7.2 format. During this migration, Studio automatically creates the appropriate data elements (characteristics) and references (from specifications) to those data elements.

Characteristic Renaming Rules

To support W3C XML Attribute Names for Namespace Declaration standards, characteristic naming rules were changed for UIM 7.2. The standard says that a valid NCName can be of the format:

```
NCName ::= (Letter | '_') (NCNameChar)
NCNameChar ::= Letter | Digit | '.' | '-' | '_' | CombiningChar | Extender
```

There are certain special characters that are not allowed in the name. Periods (.) and hyphens (-) are not allowed at the beginning of a name, but are allowed in the remainder. Although periods and hyphens are allowed in the body of a name, Oracle recommends that they not be used at all.
To conform to the naming rules, the cartridge migration process replaces all spaces, hyphens, and special characters with an underscore character (_).

Special characters include:
~ ! @ # % ^ & * ( ) - + = { } [ ] : " ; ' < > ? , . / | 

If the name starts with a number, an underscore “_” is added to the beginning of the name and the number is retained as the second character of the name. If the resulting name already exists in the workspace, an underscore and a number are appended to the name. Numbering starts at 1 and increments until a unique name can be generated.

Table 3–1 illustrates some renaming examples.

<table>
<thead>
<tr>
<th>Characteristic name before migration</th>
<th>Characteristic name after migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>max speed</td>
<td>max_speed</td>
</tr>
<tr>
<td>8port</td>
<td>_8port</td>
</tr>
</tbody>
</table>

Cartridge Migration Process Overview

This section provides an overview of the cartridge migration process, which is part of the larger process of updating the database. For step-by-step instructions, see UIM System Administrator’s Guide.

Figure 3–1 illustrates the update process. Cartridge migration is the step at the bottom of the center column.

Figure 3–1 Upgrade Process Overview
You can optionally use the `db_migration_report.properties` file generated by the database upgrade utility as input to the cartridge migration. This file identifies invalid characteristic names and provides updated ones based on the naming rules mentioned in "Characteristic Renaming Rules".

If you use the file from the upgrade utility, Design Studio updates it with any modifications you made during cartridge migration. If you do not use this file from the database utility, Design Studio generates its own file. In either case, you use the file after the cartridge migration as part of the database upgrade. See the database upgrade chapter in *UIM Installation Guide* for more information.

All cartridges requiring migration must be imported into the same Design Studio workspace. (If the cartridges are not imported into the same workspace, modified characteristic names may not be unique among all your cartridges.) In addition, two new required cartridges must be in the workspace during migration. These required cartridges are `ora_uim_mds` and `ora_uim_model`. (These cartridges are required any time you work in Design Studio on cartridges for UIM 7.2.)

To migrate cartridges, you must be working in the Design Studio Design perspective. When you open or import a pre-7.2 cartridge, Design Studio recognizes the cartridge as requiring an upgrade and displays the Studio Project Upgrade window which starts the migration process.

The Project Upgrade window shows all the cartridges that have been opened or imported that require an upgrade. Clicking the **Next** button cycles through the projects in the list and displays their characteristic information. The **Former Characteristic** column contains the name of the original characteristic, the **New Characteristic** column displays the new name if one is required. This column is blank if a new name is not required. The **Display Name** column contains the name of the display label. The screen also displays the total number of characteristic name changes, the characteristic references moving to the Data Dictionary, and the number of specification-characteristic references that are changing. You can change a proposed characteristic name by entering text in the **New Characteristic** column. The display label name is not modifiable.

### Cancellation During Migration

If the **Cancel** button is selected anytime after the finish has been selected, the Project Upgrade window will close and the Inventory projects will remain open in the workspace. The projects that were being migrated will not be processed completely and will have to be manually closed. Removing and reloading of the cartridges that were cancelled during migration would be recommended since the state of the cartridges being migrated is not known.

---

**Note:** If the project being migrated is kept open after a cancellation, the cartridge has not been migrated. The cartridge will be in an undetermined migrated state.

---

### Manual Reset of Layouts

Layout information must be reset manually after cartridge migration. Elements selected for display in the prior release must be reselected in the new version. The order of display must also be reset.
Changing JDOQL to JPQL on Query Characteristics

For 7.2, queries must be modified to use JPQL syntax. For example this JDOQL query from a previous release:

select name from Place where specification.name == 'Country'

must be converted to JPQL as follows:

select o.name from Place o where o.specification.name = 'Country'

JDOQL must also be changed in other areas, such as rule sets and custom Java code. See "Changing JDOQL to JPQL" for additional details.

Migrating Custom Rule Sets

The following sections provide information about the changes required to migrate rule sets for UIM 7.2.

Packaging Changes

Ruleset classes are organized into a different package structure in UIM 7.2. In the previous release the package was:

package com.metasolv.rules

For UIM 7.2, the package is:

package oracle.communications.inventory.rules

Import Statement

The ExtensionPointRuleContext class has been organized into a different package structure. The import statement for this class therefore be modified as listed below.

Pre-7.2:
import com.metasolv.extensibility.extension.util.ExtensionPointRuleContext;

7.2
import oracle.communications.inventory.extensibility.extension.util.\
ExtensionPointRuleContext;

Use InventoryConfigurationVersion as a Parameter in Rulesets

Change ServiceConfigurationVersion to InventoryConfigurationVersion

Pre-7.2
rule "Allocation of l2vpn Vll service configuration"
salience 2
when
then
log.debug("", new Object[] {" L2VPN_ALLOCATE_ALL_VLL_SERVICE started." });
Map methodHandler = new HashMap();
methodHandler.put(L2VpnConstants.RESOURCE_VLL_SERVICE_ID,"true");
L2VpnManager manager = L2VpnHelper.makeVpnManager();
if (svcConVers != null)
manager.allocateAllOnVLLService(methodHandler);
end
7.2
rule "Allocation of l2vpn Vll service configuration"
salience 2
when
    svcConVers : InventoryConfigurationVersion()
then
    log.debug("", new Object[] {" L2VPN_ALLOCATE_ALL_VLLSERVICE started." });
    Map methodHandler = new HashMap();
    methodHandler.put(L2VpnConstants.RESOURCE_VLL_SERVICE_ID,"true");
    L2VpnManager manager = L2VpnHelper.makeVpnManager();
    if (svcConVers != null)
        manager.allocateAllOnVLLService((ServiceConfigurationVersion)svcConVers, methodHandler);
end

Connect if the Entity is Stale

If an entity instance in a ruleset parameter is detached, you must use the connect() method to the instance.

For example:

```java
public List<ServiceConfigurationVersion> validateVLLService(
    Collection<ServiceConfigurationVersion> serviceConfigurationVersion)
throws Exception
//-------------------------------------------------------
{
    List<ServiceConfigurationVersion> returnedConfigurationVersions =
    new ArrayList<ServiceConfigurationVersion>();

    for (ServiceConfigurationVersion configVer : serviceConfigurationVersion)
    {
        configVer = persistenceManager.connect(configVer); // needs to be added for a detached configuration version.
        L2VpnVllServiceValidation l2VpnServiceValidation = new L2VpnVllServiceValidation();
        if ("PENDING_CANCEL".equals(configVer.getConfigState().getValueAsString())|
            "PENDING_DISCONNECT".equals(configVer.getService().getAdminState().getValueAsString())){
            returnedConfigurationVersions.add(configVer);
            continue;
        }else{
            l2VpnServiceValidation.validate(configVer);
            returnedConfigurationVersions.add(configVer);
        }
    }
    return returnedConfigurationVersions;
}
```
Cartridge Deployment

Note: If you have an After Rule on a Create API, the input parameter received by the rule is still the transient object and not the persistent object that was created in the API. In such scenarios, you can access the newly created persistent object from the ExtensionPointRuleContext by invoking the getReturnValue method. As stated above, in all other cases connecting the input parameters gives you the handle to the input parameter with all the changes applied by the API.

See "Migrating Custom Java Code" for information about converting the rest of the Java code.

Cartridge Deployment

The way you deploy cartridges in UIM has been changed for release 7.2. has been changed. The Cartridge Management page of UIM has been removed. All cartridges must now be deployed using the standalone Cartridge Deployer Tool or from a Studio Environment project in Design Studio. See UIM System Administrator’s Guide for information about using the Cartridge Deployer Tool. See Design Studio online Help for information about deploying from Design Studio.
This chapter provides information about issues relating to the database migration required for upgrading to Oracle Communications Unified Inventory Management (UIM) 7.2. For information about running the database migration utility, see UIM Installation Guide.

Database Upgrade Reports

The first step in upgrading the database is to run the upgrade utility in Report mode. Report mode generates two reports in the Database Upgrade Home directory:

- The *db_migration_report.properties* file lists any characteristic names that have been identified as having a character that is not valid or is in a position that is not valid. If the report is empty, all characteristics in the database follow the naming conventions for this release. If there are characteristics that require modification, their original and modified names are listed. See “Characteristic Renaming Rules” for information about how characteristic names are changed. See the database upgrade chapter in UIM Installation Guide for more information about the cartridge migration.

- The *SEQUENCE_Table_Update_Report.txt* file lists any invalid lengths in context names used to create native ORACLE sequences that replace existing table sequences. This report also displays any values that have to be manually changed by a database administrator. If a sequence is identified as having a problem, the database upgrade cannot continue proceed until the sequence has been corrected. See “SEQUENCE Table Migration” for more information.

SEQUENCE Table Migration

UIM includes a sequence mechanism for generating numbers that are unique within some context. For example, a sequence could be used to generate unique identifiers for devices manufactured by a particular vendor, such as Cisco. A sequence with a context of Cisco would be used to generate a unique identifier for each Cisco device.

A sequence value is requested for a given context using the UIM Sequence Generator API. A sequence can optionally be defined by a sequence specification which defines the minimum value, maximum value, and increment by value for the sequence.

The Sequence Generator API can be called from extension logic to generate unique numeric values for ad hoc purposes. The sequence mechanism is also used internally by UIM to automatically generate ID values for entities that have the ID attribute. These entities are defined by entity specifications that define whether or not the ID value is automatically generated. For more information about the UIM Sequence Generator interface, see UIM Information Model Reference.
Prior to release 7.2, the SEQUENCE table in UIM support this feature. In UIM 7.2, the
SEQUENCE table is dropped, and the rows are replaced by Oracle native sequences.
In some cases the value of the CONTEXT column on the SEQUENCE table is used
during data migration to generate the name of the Oracle sequence that replaces the
SEQUENCE row. Because the maximum length of the Oracle sequence name is 30
characters, there may be cases where the CONTEXT value must be shortened on a
SEQUENCE row prior to running the database migration. Any references to the
context value in extension code must also be changed to the new value. The Database
Upgrade utility includes a report (SEQUENCE_Table_Update_Report) that can be run
prior to the database upgrade to identify any SEQUENCE rows with a CONTEXT
value that must be shortened.

The sections below describe ad hoc sequence generation and automatic ID generation
scenarios and provide details of how the SEQUENCE data is migrated for each
scenario.

Ad Hoc Sequence Generation

Ad hoc sequence generation refers to generating unique numeric values for ad hoc
purposes using the Sequence Generator API in extension logic. There are two types of
ad hoc sequences that can be generated: specification-based sequences and
context-based sequences. These sequence types are described in the following sections.
Note that these types of ad hoc sequences may require the CONTEXT value on the
SEQUENCE row to be shortened prior to the database migration.

Specification-based Sequence

A specification-based sequence is used to generate a sequence within a specified
context, where the sequence is defined by a sequence specification. A sequence
specification is defined in Design Studio. It defines the minimum value, maximum
value, and increment value for the sequence.

A specification-based sequence is requested by invoking the next(String
sequenceSpecName, String context) method on the Sequence Generator interface,
where sequenceSpecName is the name of the sequence specification. The method returns
the next sequence value for the combination of sequence specification name and
context.

Prior to the UIM 7.2 release, the first time a sequence was requested for the
combination of sequence specification name and context, a SEQUENCE row was
created that stored the context value, the sequence specification, and the last generated
value for the sequence. Each subsequent request for a sequence value for the
combination of sequence specification name and context returned a value equal to the
last generated value plus the increment by value defined on the sequence specification.
Each request results in an update to the CURRENTVALUE on the SEQUENCE row.

In the 7.2 database migration, the SEQUENCE row that represents a
specification-based sequence is replaced by an Oracle sequence. The database
migration identifies a SEQUENCE row as a specification-based sequence when the
SPECIFICATION column is populated with the JDOID of a
SEQUENCESPECIFICATION row, and the CONTEXT value does not equal the name
of an existing Entity Identification specification (a CONTEXT value equal to a existing
Entity Identification specification identifies a sequence used for Custom ID
Generation). The increment value and maximum value of the Oracle sequence will be
set to the INCREMENTVALUE and MAXVALUE defined on the
SEQUENCESPECIFICATION row of the sequence specification.
The name of the Oracle native sequence will be generated with the format `CONTEXT_ SequenceSpecification ENTITYID` where `CONTEXT` is the given context value, and `SequenceSpecification JDOID` is the JDOID (primary key) value on the SEQUENCESPECIFICATION row for the sequence specification. For example, if the context value is Cisco, and the sequence specification JDOID is 1724995, the Oracle sequence name will be generated as Cisco_1724995.

The maximum length for the CONTEXT portion of the name for a specification-base Oracle native sequence is 10 characters. This constraint is due to the fact that the maximum length of an Oracle native sequence name is 30 characters, and the JDOID, (defined as NUMBER(19)) and the underscore take up 20 of the 30 characters. If the CONTEXT value on an existing specification-based SEQUENCE row exceeds 10 characters, the CONTEXT value must be changed to a length of 10 or less prior to the database migration. Any references to the context value in extension code that invokes the Sequence Generator API must also be changed to the new value or the method calls will result in runtime errors. The database upgrade utility includes a report (SEQUENCE_Table_Update_Report) that can be run prior to the database upgrade to identify the SEQUENCE rows for specification-based sequences that must be modified.

**Context-based Sequence**

A context-based sequence is used to generate a sequence within a specified context, where the sequence is not defined by a sequence specification.

A context-based sequence is requested by invoking the `next(String context)` method on the Sequence Generator interface. The method returns the next sequence value for the given context. The starting sequence value is 1 and increments by 1 with each request for a new value.

Prior to the UIM 7.2 release, the first time a sequence was requested for the context, a SEQUENCE row was created that stored the context value and the last generated value for the sequence. Each subsequent request for a sequence value for the context returns a value equal to the last generated value increment by 1, and results in an update to the current value on the SEQUENCE row.

In the 7.2 database migration, the SEQUENCE row that represents a context-based sequence is replaced by an Oracle sequence. The database migration identifies a SEQUENCE row as a context-based sequence when the SPECIFICATION column is not populated, and the CONTEXT value does not equal a fully qualified implementation class name of a UIM entity type, such as “com.metasolv.impl.entity.LogicalDeviceDAO” (a CONTEXT value equal to a fully qualified implementation class name identifies a sequence used for Default ID Generation).

The name of the Oracle native sequence will be set to the CONTEXT value on the SEQUENCE row. Because the maximum length of an Oracle sequence name is 30 characters, the context value for a context-based sequence cannot exceed 30 characters. If the CONTEXT value on an existing context-based SEQUENCE row exceeds 30 characters, the CONTEXT value must be changed to a length of 30 or less prior to the database migration. Any references to the context value in extension code that invokes the Sequence Generator API must also be changed to the new value, or the method calls will result in runtime errors. The database upgrade utility includes a report (SEQUENCE_Table_Update_Report) that can be run prior to the database upgrade to identify the SEQUENCE rows for context-based sequences that must be modified.
Automatic ID Generation

Some entities in UIM have an ID attribute that represents a business identifier for the entity. The attribute value is required, and the entity specification defines whether the ID value must be manually entered or is automatically generated by the application. UIM internally uses the sequence mechanism to automatically generate ID values. There are two scenarios of automatic ID generation: Custom ID generation and Default ID generation. The SEQUENCE table migration for these scenarios is described in the sections below. For more information about automatic ID generation, see the Entity Identification Pattern section of the UIM Information Model Reference document.

Note that SEQUENCE rows that support automatic ID generation are migrated to Oracle sequences without the need to shorten the CONTEXT value prior to running the database migration.

Custom ID Generation

Custom ID generation occurs when an entity is created from an entity specification that declares that the ID value is automatically generated and the entity specification is associated with an Entity Identification specification. The Entity Identification specification defines a format for the ID value that can have a predefined prefix and/or suffix value. The prefix and suffix are combined with a generated sequence value to form the ID value in the form of prefix + sequence value + suffix. The Entity Identification specification is related to a sequence specification that defines the minimum value, maximum value, and increment by value for the sequence.

Prior to release 7.2, a dedicated SEQUENCE row was used to generate the sequence value portion of the ID values for entities created from entity specifications that are associated to the same Entity Identification specification. The CONTEXT column on the SEQUENCE row is set to the name of the Entity Identification specification, and the SPECIFICATION column is populated with the JDOID value of the SEQUENCESPECIFICATION row for the sequence specification.

The 7.2 data migration replaces each SEQUENCE row used for custom ID generation with an Oracle native sequence named ID_JDOID, where JDOID is the internal primary key for the Entity Identification specification instance. For example, the Oracle sequence that replaces the SEQUENCE row for the sequence used on an Entity Identification specification with an JDOID of 1340001 would be named ID_1340001. These Oracle sequence names will all be within the maximum length of 30 characters, so there will be no need to modify the SEQUENCE rows prior to the database upgrade.

Default ID Generation

Default ID generation occurs when an entity is created from an entity specification that declares that the ID value is not automatically generated and the entity specification is not associated with an Entity Identification specification. For each entity type, such as Logical Device, a dedicated sequence is used to generate ID values for the entity type. Each sequence is represented by a row in the SEQUENCE table where the CONTEXT value equals the fully qualified implementation class name for the entity type. For example, for the Logical Device class, the CONTEXT value is com.metasolv.impl.entity.LogicalDeviceDAO.

The 7.2 data migration replaces each SEQUENCE row used for default ID generation with an Oracle native sequence named ENTITY TYPE_SEQ. For example, the Oracle sequence that replaces SEQUENCE row for the Logical Device sequence is named LOGICALDEVICE_SEQ. These Oracle sequence names are all within the maximum length of 30 characters, so there will be no need to modify the SEQUENCE rows prior to the database upgrade.
This chapter describes changes that you need to migrate custom code written for previous releases of Oracle Communications Unified Inventory Management (UIM) to UIM 7.2.

### Class Names and Package Changes

Some framework classes were renamed or repackaged for UIM 7.2. You must change the references of these classes and their imports for your code to compile.

Table 5–1 lists classes that were either renamed or repackaged for UIM 7.2.

<table>
<thead>
<tr>
<th>7.1</th>
<th>7.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.metasolv.api.InventoryHelper</td>
<td>oracle.communications.platform.persistence.PersistenceHelper</td>
</tr>
<tr>
<td>oracle.communications.inventory.api.framework.persistence.Finder</td>
<td>oracle.communications.platform.persistence.Finder</td>
</tr>
<tr>
<td>javax.jdo.JDOHelper</td>
<td>Dropped. These methods are now available on the PersistenceManager.</td>
</tr>
<tr>
<td>oracle.communications.inventory.api.TypeRegistryBean</td>
<td>oracle.communications.platform.persistenceTypeRegistry</td>
</tr>
<tr>
<td>oracle.communications.inventory.api.Persistent</td>
<td>oracle.communications.platform.persistence.Persistent</td>
</tr>
</tbody>
</table>

- The factory manager InventoryHelper has been replaced with PersistenceHelper. Use PersistentHelper class to get an instance of the Finder or a Manager.
- Most methods in JDOHelper have been moved to the PersistenceManager. The following are some of the commonly used JDOHelper methods which have changed
  - Replace JDOHelper.makeDirty(java.lang.Object p1, java.lang.String p2) with PersistenceManager.makeDirty(Persistent entity).
  - Replace JDOHelper.getObjectId(java.lang.Object p1) with PersistenceManager.getPomsObjectId(Persistent entity).
  - Replace javax.jdo.PersistenceManager JDOHelper.getPersistenceManager(java.lang.Object p1) with javax.persistence.EntityManager PersistenceManager.getPersistenceManager(Persistent entity).
- Replace JDOHelper.isDirty(java.lang.Object p1) with PersistenceManager.isDirty(Persistent entity).
- Replace JDOHelper.isTransactional(java.lang.Object p1) with PersistenceManager.isTransactional(java.lang.Object obj).
- Replace JDOHelper.isPersistent(java.lang.Object p1) with PersistenceManager.isPersistent(java.lang.Object obj).
- Replace JDOHelper.isNew(java.lang.Object p1) with PersistenceManager.isNew(java.lang.Object obj).
- Replace JDOHelper.isDeleted(java.lang.Object p1) with PersistenceManager.isDeleted(java.lang.Object obj).

All entities extend the Persistent interface; the following methods have changed on this interface. Table 5–2 lists the old and new methods.

<table>
<thead>
<tr>
<th>Table 5–2 Changed Methods on Persistence Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>getJdoId()</td>
</tr>
<tr>
<td>getJdoClass()</td>
</tr>
<tr>
<td>getJdoVersion()</td>
</tr>
</tbody>
</table>

## Returned Collections

Accessory methods on generated entities now return a Collection interface instead of a concrete Collection class. A compilation error occurs if the custom code is written using the concrete class as the return type.

For example, on the generated Equipment entity

```java
java.Util.ArrayList getPhysicalPorts()
```

has changed to

```java
java.Util.List<oracle.communications.inventory.api.entity.PhysicalPort> getPhysicalPorts()
```

To address this issue you need to change the return type to be the appropriate Collection interface or cast the return as the concrete Collection class.

For example, custom code such as

```java
ArrayList<PhysicalPort> ppList = eq.getPhysicalPorts()
```

causes a compilation error. The error can be fixed by changing the code in one of two ways:

- ```java
   List<PhysicalPort> ppList = eq.getPhysicalPorts()
   ```
- ```java
   ArrayList<PhysicalPort> ppList = (ArrayList<PhysicalPort>) eq.getPhysicalPorts()
   ```

## Finder

This class is a wrapper that enables users to write queries in custom code. Because of the changes in the persistence framework, the methods in this class were modified significantly. Backward compatibility was maintained as much as possible, but was not feasible in all situations.
The following methods were changed. Refer to the Javadoc for details.

### Table 5–3  Changed Finder Methods

<table>
<thead>
<tr>
<th>7.1</th>
<th>7.2 Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>find(String queryExpression)</td>
<td>findByJPQL(String queryExpression)</td>
</tr>
<tr>
<td>find(Class&lt; E &gt; candidateType, String [] paramNames, String filter, Object [] params)</td>
<td>find(Class&lt; E &gt; candidateType, String filter, String [] paramNames, Object [] params)</td>
</tr>
<tr>
<td>find(Class&lt; E &gt; candidateType, String paramDecl, String filter, Object [] params)</td>
<td>find(Class&lt; E &gt; candidateType, String filter, String [] paramNames, Object [] params)</td>
</tr>
<tr>
<td>addEagerFetchFields()</td>
<td>addEagerFetchFields(String... fields)</td>
</tr>
<tr>
<td>setFilter(String filter)</td>
<td>setJPQLFilter(String filter)</td>
</tr>
<tr>
<td>setSqlString(String sqlString)</td>
<td>findByNativeSQL(String sql)</td>
</tr>
<tr>
<td>setQueryLanguage(String queryLanguage)</td>
<td>findByNativeSQL(Class candidateType, String sql)</td>
</tr>
<tr>
<td>setOrdering(String ordering)</td>
<td>addDescendingOrdering(String... attributeName)</td>
</tr>
<tr>
<td>setGroupingList(List names)</td>
<td>Dropped. Use Group By clause from JPQL</td>
</tr>
<tr>
<td>declareParameters(String paramDeclls)</td>
<td>Dropped. Not needed.</td>
</tr>
<tr>
<td>declareImports(String importDeclls)</td>
<td>Dropped. Not needed.</td>
</tr>
</tbody>
</table>

**Finder.findByEntity**

The semantics of the findByEntity method has changed for the UIM 7.2 release, although the signature has remain the same.

In previous releases, the `findByEntity(object, attribute 1, attribute2, ...)` method returned only objects that matched the input object on all the attributes passed in. It ignored all other attributes on the input object.

In UIM 7.2, the method returns objects that match on all non-null attributes in the input object in addition to the input attributes.

**Finder.setRange**

In previous releases, a request to bring a range of results from a starting row number to an ending row number, did not include the ending row number in the result set. In UIM 7.2, a similar request also returns the row with the ending number.

**InventoryFinder**

InventoryFinder is a new class added in this release. This class is a subclass of the Finder class. The factory method

```java
public static oracle.communications.platform.persistence.Finder makeFinder
```

on the PersistenceHelper returns an instance of the InventoryFinder, even though the return type is Finder.

To access the methods on the InventoryFinder, you must cast the returned reference to InventoryFinder. The following sample code invokes the getTotalCounts method on the InventoryFinder class.
Changing JDOQL to JPQL

If your code creates a Finder using JDOQL, it must be converted to the JPQL query language. See the EclipseLink documentation for information about JPQL syntax.

Some common changes from JDOQL to JPQL are:

- Replace the `contains` filter clause with `like`.
- Replace `&&` with `AND`.
- Replace `||` with `OR`.
- Replace `==` with `=`.
- The parameters to the query have to be prefixed with colon. For example, replace `param` with `:param`.
- The query selection must be aliased. For example, a JDOQL statement like
  
  ```
  select name from specification where id == 'test'
  ```

  can be converted to

  ```
  select o.name from specification o where o.id = 'test'
  ```

Eager Fetch Changes

In Kodo, a large number of the relationships of an entity may need to be fetched before the transaction is committed. If this is not done and this relationship is accessed on the detached object, the caller is returned null. This can lead to confusion as the caller has no way of knowing whether the relationship is not fetched in the transaction or if the relationship is actually null.

For example, when an Equipment object (for example `equip`) is retrieved and the transaction is committed, the `equip` object is detached. If the caller invokes `equip.getPhysicalPorts()`, this method would return null. The caller does not know whether the entity has ports or not.

To avoid this problem, the following needed to be done:

- `equip.getPorts()` had to be executed before committing the transaction.
- Do an eager fetch on the relationship `physicalPort`. when finding the object using a Finder. This caused the generated SQL to do an outer join to the PhysicalPort table, thereby populating the ports on the Equipment object.

EclipseLink supports lazy loading of relationships and primitive attributes. This means that when a relationship is accessed on a detached object and the relationship has not yet been accessed, the relationship is populated. Due to this enhancement in EclipseLink, we do not have to eager fetch relationships.

EclipseLink does support eager fetching, which can be set on the Finder. There are three types of eager fetching supported in EclipseLink: Batch, Join and Outer join. Refer to the EclipseLink documentation and UIM Developer’s Guide for more details on these fetch types.

In the previous release, the find APIs provided by UIM used eager fetch for the most common relationships. This was difficult to maintain because it was not possible to guess which relationships the caller was interested in accessing. But it was not...
practical to eager fetch all possible relationships because of performance issues. To avoid these problems, the UIM 7.2 APIs no longer eager fetches relationships.

**Consumer Model Changes**

For performance reasons, the relationship between resources and consumers was made unidirectional in 7.2. This relationship was bidirectional in previous releases. After this change, the method getConsumers on a Consumable resource is no longer generated.

To get the consumers (assignments, conditions, reservations) for a resource, the following APIs should be used

- `public Collection<Consumer> getConsumers(ConsumableResource consumableResource) throws ValidationException on the ConsumerManager`
- `public Collection<Assignment> findAssignment(AssignmentSearchCriteria assignmentSearchCriteria) throws ValidationException on the AssignmentManager`
- `public Collection<? extends Reservation> findReservations(ReservationSearchCriteria criteria) throws ValidationException; on the ReservationManager`
- `public Collection<? extends Condition> findConditions(ConditionSearchCriteria criteria) throws ValidationException; on the ConditionManager`

Please refer to the model doc for the description of the Consumer pattern and the Javadoc for these methods.

**Entity Life Cycle Listener**

The EntityLifeCycleEventListenerImpl listener class has been replaced by the InventoryEntityLifeCycleEventListener class in 7.2. In the previous release, you set extension points on methods on the EntityLifeCycleEventListenerImpl class. In UIM 7.2, you set extension points on the InventoryEntityLifeCycleEventListener class.

The following table lists the replacements methods. Note that these changes are dictated by changes in the Persistence framework. Refer to the EclipseLink documentation for explanations on when these events are triggered.

<table>
<thead>
<tr>
<th>Table 5-4 Entity Life Cycle Listener Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
</tr>
<tr>
<td>preCreate</td>
</tr>
<tr>
<td>postCreate</td>
</tr>
<tr>
<td>preUpdate</td>
</tr>
<tr>
<td>postUpdate</td>
</tr>
<tr>
<td>preDelete</td>
</tr>
<tr>
<td>postDelete</td>
</tr>
<tr>
<td>postLoad</td>
</tr>
</tbody>
</table>

In the previous release, users could configure the life cycle listener by setting certain properties in the `UIM_Home/config/resource/event.properties` file. This configuration
is no longer available at the system level. The following subsections guide the user on how to configure the entity life cycle listeners in the 7.2 release.

Disabling /Enabling life cycle listeners.

In the 7.1 release, setting the following properties to false in the events.properties file disabled all the life cycle listeners for all entities.

create.all=false
read.all=false
update.all=false
delete.all=false

This was the default behavior of the system.

To enable the life cycle listeners for all the entities, users could turn all the above to true or selectively turn a few on. For example, user could change the create.all to true and leave the other to false.

In the 7.2 releases, life cycle listeners can be configured by writing extension code on the following methods.

- prePersist()
- postLoad()
- preUpdate()
- preRemove()

Entity-specific configuration

In the 7.1 release, users could set the following configuration in the events.properties file

oracle.communications.inventory.api.entity.Equipment=create,update

to enable entity life cycle configuration callback for a specific entity. In this example, only Equipment entities are configured for create and update events.

In the 7.2 release, configuration per entity is not supported through the properties file. This functionality can be achieved by using the extension points and creating custom code. The following extension points must be used to achieve this functionality:

- prePersist()
- preUpdate()

The following is a sample implementation of the extension code on the prePersist() method to control the custom logic to be executed only for an Equipment entity and not take any actions for the others.

```java
@PrePersist
public void prePersist(Object object) {
    super.prePersist(object);
    if (log.isDebugEnabled()) {
        log.debug("", "InventoryEntityLifeCycleListener In prePersist: " +
                   object);
    }
    if(object instanceof Equipment) {
        // custom logic goes here
    }
    // use following condition to call custom logic in lifecycle methods only
```
for certain specification
    else if (object instanceof Equipment &&
            "SampleEquipmentSpec".equals(((Equipment)object).getSpecification().getName()))
    {
        // custom logic for specification based lifecycle listener
    }
}
This document outlines changes that are required to migrate Web services to Oracle Communications Unified Inventory Management 7.2.

Framework Changes

Figure 6–1 and Figure 6–2 illustrate the structure of the Web Service Framework in UIM 7.1.2 and 7.2. Table 6–1 provides a side-by-side comparison of names in the two releases.

Figure 6–1  UIM 7.1.2 Web Service Framework

```xml
oracle.communications.inventory.webservice.framework.AdapterRoot

getTransaction() : UserTransaction
commitOrRollback( userEnvironment : UserEnvironment, ut : UserTransaction )
commitOrRollback( transVal : TransactionValue )
commit( transVal : TransactionValue )
rollback( transVal : TransactionValue )
startTransaction() : TransactionValue

+logEntity( log : Log, entity : Object, message : String )
+logEntities( log : Log, entities : Object [0..*], message : String )
-logSingleEntity( log : Log, entity : Object, message : String )
+startUserEnvironment( requestContext : RequestContext ) : UserEnvironment
+endUserEnvironment( userEnvironment : UserEnvironment, inventoryResponseType : InventoryResponseType )
+checkFeedbackProvider( userEnvironment : UserEnvironment, requestContext : RequestContext )
-getBusinessInteraction( bid : String ) : BusinessInteraction
+copyFeedBackMessages( feedbackProvider : FeedbackProvider, inventoryResponseType : InventoryResponseType )
-getRequestPolicy() : RequestPolicy
```

```xml
oracle.communications.inventory.webservice.adapter.<Entity>Adapter-7.1.2
```
Figure 6–2  UIM 7.2 Web Service Framework

Table 6–1  Name Changes

<table>
<thead>
<tr>
<th>7.1.2</th>
<th>7.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.metasolv.api.util.Util</td>
<td>oracle.communications.inventory.api.util.Util</td>
</tr>
<tr>
<td>com.metasolv.api.InventoryHelper</td>
<td>oracle.communications.platform.persistence.PersistenceHelper</td>
</tr>
<tr>
<td>com.metasolv.api.framework.logging.impl.FeedbackProviderImpl</td>
<td>oracle.communications.inventory.api.framework.logging.impl.FeedbackProviderImpl</td>
</tr>
<tr>
<td>oracle.communications.inventory.webservice.framework.AdapterRoot</td>
<td>oracle.communications.inventory.webservice.framework.InventoryAdapterRoot</td>
</tr>
<tr>
<td>oracle.communications.inventory.webservice.framework.TransactionValue</td>
<td>oracle.communications.inventory.webservice.framework.InventoryTransactionValue</td>
</tr>
<tr>
<td>com.metasolv.api.InventoryHelper</td>
<td>oracle.communications.platform.persistence.PersistenceHelper</td>
</tr>
<tr>
<td>com.metasolv.api.framework.logging.impl.FeedbackProviderImpl</td>
<td>oracle.communications.inventory.api.framework.logging.impl.FeedbackProviderImpl</td>
</tr>
</tbody>
</table>
Table 6–1  (Cont.) Name Changes

<table>
<thead>
<tr>
<th>7.1.2</th>
<th>7.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle.communications.inventory.api.TypeRegistry</td>
<td>oracle.communications.platform.persistence.TypeRegistry</td>
</tr>
<tr>
<td>oracle.communications.inventory.api.framework.persistence.CriteriaItem</td>
<td>oracle.communications.platform.persistence.CriteriaItem</td>
</tr>
<tr>
<td>oracle.communications.inventory.api.framework.persistence.CriteriaOperator</td>
<td>oracle.communications.platform.persistence.CriteriaOperator</td>
</tr>
<tr>
<td>oracle.communications.inventory.api.framework.persistence.Finder</td>
<td>oracle.communications.platform.persistence.Finder</td>
</tr>
</tbody>
</table>

Changes to the Web Services Framework Schema

This section lists changes the Web Service Framework Schema

**FaultRoot**

The FaultRoot.xsd file was added.

```xml
<schema

targetNamespace="http://xmlns.oracle.com/communications/framework/webservice/fault/v1-0" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:appfault="http://xmlns.oracle.com/communications/framework/webservice/fault/v1-0" elementFormDefault="qualified">
  <complexType name="ApplicationFaultType">
    <sequence>
      <element name="errorCode" type="string" nillable="false"/>
      <element name="errorMessage" type="string" nillable="true" minOccurs="0"/>  
      <element name="errorContext" type="string" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</schema>
```

**InventoryFaults.xsd**

You must add the namespace appfault and import FaultRoot.xsd.

**7.1.2**

```xml
<xs:schema

targetNamespace="http://xmlns.oracle.com/communications/inventory/webservice/fault" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:invfault="http://xmlns.oracle.com/communications/inventory/webservice/fault"
>  
<xs:import namespace="http://www.w3.org/2001/XMLSchema"/>
</schema>
```

**7.2**

```xml
<xs:schema

targetNamespace="http://xmlns.oracle.com/communications/inventory/webservice/fault" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:invfault="http://xmlns.oracle.com/communications/inventory/webservice/fault"
xmlns:appfault="http://xmlns.oracle.com/communications/framework/webservice/fault/
```
The elements `errorMessages` and `rootStackTrace` were removed from the complex type `InventoryFaultType`. They were replaced with the element `fault`. The element `fault` is of type `ApplicationFaultType`.

### 7.1.2

```xml
<xs:complexType name="InventoryFaultType">
  <xs:sequence>
    <xs:element name="errorMessages" type="xsd:string" .../>
    <xs:element name="rootStackTrace" type="xsd:string" .../>
  </xs:sequence>
</xs:complexType>
```

```xml
<xs:complexType name="InventoryFaultType">
  <xs:sequence>
    <xs:element name="fault" type="appfault:ApplicationFaultType"/>
  </xs:sequence>
</xs:complexType>
```

### type-mapping.xml

The `fault v1-0` package/namespace mapping was added.

```xml
<package-mapping>
  <package-type>oracle.communications.inventory.webservice.framework.fault</package-type>
  <namespaceURI>http://xmlns.oracle.com/communications/framework/webservice/fault/v1-0</namespaceURI>
</package-mapping>
```

### Inventory.xsdconfig

The `fault v1-0` namespace/package mapping was added.

```xml
<package-mapping>
  <package-type>oracle.communications.inventory.webservice.framework.fault</package-type>
  <namespaceURI>http://xmlns.oracle.com/communications/framework/webservice/fault/v1-0</namespaceURI>
</package-mapping>
```

### InventoryCommon.xsd

You must add the namespace `appfault` and import `FaultRoot.xsd`.

### 7.1.2

```xml
<xs:schema targetNamespace="http://xmlns.oracle.com/communications/inventory/webservice/commo"
Changes to the Web Services Framework Schema

The requestContext element was removed from the complex type InventoryRequestType.

7.1.2

<xs:complexType name="InventoryRequestType">
  <xs:sequence>
    <xs:element name="header" type="xsd:string" ..../>
    <xs:element name="requestContext" type="invcom:RequestContext" ..../>
  </xs:sequence>
</xs:complexType>

7.2

<xs:complexType name="InventoryRequestType">
  <xs:sequence>
    <xs:element name="header" type="xsd:string" ..../>
  </xs:sequence>
</xs:complexType>

The elements requestContext and messageConfirmation were removed from the complex type InventoryResponseType. Also, the type for the element errors was changed from string to ApplicationFaultType. Note that the type ApplicationFaultType is defined in the newly imported schema file FaultRoot.xsd.

7.1.2

<xs:complexType name="InventoryResponseType">
  <xs:sequence>
    <xs:element name="requestContext" type="invcom:RequestContext" ..../>
    <xs:element name="messageConfirmation" type="xsd:string" ..../>
    <xs:element name="warnings" type="xsd:string" ..../>
    <xs:element name="errors" type="xsd:string" ..../>
    <xs:element name="messages" type="xsd:string" ..../>
  </xs:sequence>
</xs:complexType>

7.2

<xs:complexType name="InventoryResponseType">
  <xs:sequence>
    <xs:element name="requestContext" type="invcom:RequestContext" ..../>
    <xs:element name="messageConfirmation" type="xsd:string" ..../>
  </xs:sequence>
</xs:complexType>
Development Environment Changes

This section describes changes to the Web Services development environment.

Recommended Eclipse .classpath file

These are the changes made to the Eclipse .classpath file for the Reference Web service.

**Table 6–2  Eclipse Classpath File Changes**

<table>
<thead>
<tr>
<th>Version</th>
<th>Classpath Entries</th>
</tr>
</thead>
</table>
| 7.1.2   | <?xml version="1.0" encoding="UTF-8"?>  
<classpath>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="src" path="codegen/src"/>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="var" path="UIM_LIB/mslv-api-framework.jar"/>
  <classpathentry kind="var" path="UIM_LIB/mslv-core.jar"/>
  <classpathentry kind="var" path="UIM_LIB/mslv-entity.jar"/>
  <classpathentry kind="var" path="UIM_LIB/mslv-entity-framework.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-webservices-framework.jar"/>
  <classpathentry kind="var" path="WL_LIB/xbean.jar"/>
  <classpathentry kind="var" path="WL_LIB/weblogic.jar"/>
  <classpathentry kind="var" path="WL_LIB/webservices.jar"/>
  <classpathentry kind="var" path="WL_LIB/webservices.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-api-framework.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-entities.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-entities.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-managers.jar"/>
  <classpathentry kind="var" path="UIM_LIB/groupenabled_caps.jar"/>
  <classpathentry kind="var" path="UIM_LIB/com.bea.core.xml.beaxmlbeans_2.3.0.0_2-5-1.jar"/>
  <classpathentry kind="var" path="FMW_LIB/com.bea.core.xml.beaxmlbeans_2.3.0.0_2-5-1.jar"/>
</classpath>
|
| 7.2.0   | <?xml version="1.0" encoding="UTF-8"?>  
<classpath>
  <classpathentry kind="src" path="codegen/src"/>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="var" path="UIM_LIB/uim-api-framework.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-webservices-framework.jar"/>
  <classpathentry kind="var" path="POMS_LIB/platformWsFramework.jar"/>
  <classpathentry kind="var" path="POMS_LIB/platformWsFramework.jar"/>
  <classpathentry kind="var" path="POMS_LIB/platformWsFramework.jar"/>
  <classpathentry kind="var" path="POMS_PLIB/poms.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-entities.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-entities.jar"/>
  <classpathentry kind="var" path="UIM_LIB/uim-managers.jar"/>
  <classpathentry kind="var" path="UIM_LIB/groupenabled_caps.jar"/>
  <classpathentry kind="var" path="FMW_LIB/javatransactions_1.0.0.0_1-1.jar"/>
  <classpathentry kind="var" path="FMW_LIB/com.bea.core.xml.beaxmlbeans_2.3.0.0_2-5-1.jar"/>
</classpath>

For additional information on the Eclipse classpath variables, refer to the UIM Developer’s Guide.
Build Change

Need to get FaultRoot.xsd, InventoryCommon.xsd and InventoryFaults.xsd from the $UIM_HOME/lib/uim-webservices-framework.jar file.

If using the pre-7.2.0 Sample WS, the ANT build script target (get-framework-files) could be enhanced to include the FaultRoot.xsd. Example:

```xml
<target name="get-framework-files" description="Gets the framework files">
  <unjar src="${APP_LIB}/uim-webservices-framework.jar" dest="wsdl/referenceSchemas">
    <patternset>
      <include name="InventoryCommon.xsd" />
      <include name="InventoryFaults.xsd" />
      <include name="FaultRoot.xsd" />
    </patternset>
  </unjar>
</target>
```

.java File Changes

When creating a new user environment, you no longer need to give the request context as argument. For example:

7.1.2

```java
userEnvironment = 
startUserEnvironment(calculateDeltaDesignRequest.getRequestContext());
```

7.2

```java
userEnvironment = startUserEnvironment();
```

In addition:

- **Remove** `logEntity(....)` and instead use `log.info(....)` or `log.debug(....)`
- **Remove** `logException` and instead use `log.error(....)`
- **Remove** `getRequestContext` and `setRequestContext` because they were removed from the `InventoryCommon.xsd`.
- Replace `setErrors (String[])` statements with `setErrors(ApplicationFaultType[])`. If possible, it would be better to build an ApplicationFaultType and throw a new ApplicationFaultType error. For example:

```java
ApplicationFaultType af = new ApplicationFaultType();
af.setErrorCode("");
String[] errorMessages = new String[1];
errorMessages[0] = MessageResource.getMessage("ws.invalidOrderData", orderId, ....);
af.setErrorMessage(errorMessages[0]);
throw new InventoryFaultType(af);
```

- Replace `setErrors` statements like `response.setErrors(new String[] { FAILED, t.getLocalizedMessage() })` with `FeedbackUtils.copyFeedbacktoResponse(response)`
- Use the PersistenceHelper instead of the InventoryHelper. For example:

7.1.2

```java
ServiceConfigurationManager serviceConfigurationManager =
```
InventoryHelper.makeServiceConfigurationManager()

See "Migrating Custom Java Code" for information about converting the rest of the Java code.
Migrating Security Data

In previous releases of Oracle Communications Unified Inventory Management (UIM), security was handled internally by the application. In UIM 7.2, security features are now mostly handled by the application server and related tools. Security data must therefore be migrated from database tables.

**Migrating Users, Partitions and Roles**

In previous releases, user, partition, and role information was stored in the database. In UIM 7.2, these features are handled in various ways by WebLogic Server.

The following sections provide information about migrating to the new system. The examples are based on a portion of a pre-7.2 INVENTORY_USER table, shown in Figure 7-1.

![INVENTORY_USER Database Table](image)

<table>
<thead>
<tr>
<th>NAME</th>
<th>OWNER</th>
<th>USERPARTITION</th>
<th>ROLES</th>
<th>PASSWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv</td>
<td>inv</td>
<td>/</td>
<td>superuser</td>
<td>clayt0WnD0h...</td>
</tr>
<tr>
<td>joe</td>
<td>joe</td>
<td>/</td>
<td>superuser</td>
<td>clayt0Web0ch...</td>
</tr>
<tr>
<td>Mary</td>
<td>Mary</td>
<td>/user_partition</td>
<td>resourceAdmin</td>
<td>clayt0WebR0h...</td>
</tr>
<tr>
<td>John</td>
<td>John</td>
<td>/user_partition/rmpartition</td>
<td>admin</td>
<td>clayt0WnD0h...</td>
</tr>
</tbody>
</table>

**Migrating Users**

In UIM 7.2, the default authenticator stores users in the WebLogic Server Embedded LDAP. In previous releases, users were stored in a database table. When upgrading to UIM 7.2 from previous releases, all users from the INVENTORY_USER table must be migrated to the 7.2 user store. For each user defined in the table, a user has to be created using the WebLogic Server Admin Console.

You must create a user in WebLogic Server that corresponds to each of the entries in the NAME column in the table.

**Note:** As a result of the user migration process, passwords are reset for users created in WebLogic Server. Users’ previous passwords will no longer work after the migration.

See *UIM System Administrator’s Guide* for more information about how to create users.
Migrating Partitions

In UIM 7.2, partitions are defined as WebLogic Server groups. In previous releases, partitions were maintained in a database table called PARTITION. Although the behavior remains the same. Partitions are set in the PARTITION column of the entity table.

There are two groups of tasks required for migrating partitions from a previous release to UIM 7.2:

- Creating WebLogic Server Groups
- Assigning Users to Groups

See *UIM System Administrator’s Guide* for more information about these tasks.

Creating WebLogic Server Groups

To define a user partition in UIM 7.2, you must create a WebLogic Server group with a name that starts with `ora_uim_partition`.

In previous releases, partition names included slashes, which must be replaced with hash signs (#). To replace the partitions in Figure 7–1, you must create the following groups in WebLogic Server. (The root partition is supported by default; an equivalent WebLogic Server group is not required.)

- `ora_uim_partition#us_partition#tn_partition`
- `ora_uim_partition#us_partition`

Assigning Users to Groups

After creating the groups in WebLogic Server, you must assign users to them by using WebLogic Server Console. Assign users to the groups that correspond to the partitions to which they were assigned in previous releases.

For example, the users listed in Figure 7–1 must be assigned to the groups shown in Table 7–1. Because the users inv and joe were assigned to the root partition, they do not need to be associated explicitly with a group.

<table>
<thead>
<tr>
<th>User Name</th>
<th>Group Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv</td>
<td></td>
</tr>
<tr>
<td>joe</td>
<td></td>
</tr>
<tr>
<td>mary</td>
<td><code>ora_uim_partition#parentPartition</code></td>
</tr>
<tr>
<td>john</td>
<td><code>ora_uim_partition#parentPartition#childPartition</code></td>
</tr>
</tbody>
</table>

Migrating Roles

In UIM 7.2, user roles are supported using the Oracle ADF Security Framework. In previous releases, roles were maintained in a database table. Figure 7–1 show roles assigned to several users.

For role in the previous version, you must create a corresponding application role by using the Enterprise Manager (EM) Console. See *UIM System Administrator’s Guide* for more information.
Migrating Security Policies

In UIM, security policies define whether a user has permission to view or perform actions in work areas (known as pages in previous releases).

In previous releases, security policies were stored in the SECURITYPOLICY table. Now they are handled in the EM Console.

Note: Migration of security policies must be done after UIM installation. Any custom security policies defined by the customer must be migrated to 7.2. There is no automated migration path for security policies in UIM 7.2.

Migrating Authorization Policies for Work Areas

Every work area in UIM is a secured resource. A user must be authorized to view a work area. For every request, a user’s authorization policies are evaluated. Access is granted only if the user has permissions to view the work area.

Figure 7–2 shows the SECURITYPOLICY database table for a previous release of UIM.

Figure 7–2 SECURITYPOLICY Database Table

<table>
<thead>
<tr>
<th>XID</th>
<th>DESCRIPTION</th>
<th>NAME</th>
<th>ROLEREQUIRED</th>
<th>WRITEREQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>CustomObjectSummary page</td>
<td>CustomObjectSummary</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>681</td>
<td>CustomObjectSpecSearch page</td>
<td>CustomObjectSpecSearch</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>1013</td>
<td>CustomObjectSearch page</td>
<td>CustomObjectSearch</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>1121</td>
<td>CustomObjectSearchPopup page</td>
<td>CustomObjectSearchPopup</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>1261</td>
<td>CustomObjectRoleMaint page</td>
<td>CustomObjectRoleMaint</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1348</td>
<td>CustomObjectSpecSummary page</td>
<td>CustomObjectSpecSummary</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>2755</td>
<td>CustomObjectMaint page</td>
<td>CustomObjectMaint</td>
<td>(null)</td>
<td>1</td>
</tr>
<tr>
<td>3042</td>
<td>CustomObjectSpecMaint page</td>
<td>CustomObjectSpecMaint</td>
<td>(null)</td>
<td>1</td>
</tr>
<tr>
<td>3131</td>
<td>CustomObjectRoleList page</td>
<td>CustomObjectRoleList</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>3334</td>
<td>CustomObjectInventoryGroupsList page</td>
<td>CustomObjectInventoryGroupsList</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>3522</td>
<td>CustomObjectPlaceValueList page</td>
<td>CustomObjectPlaceValueList</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>3604</td>
<td>CustomObjectAssignSearchPopup page</td>
<td>CustomObjectAssignSearchPopup</td>
<td>(null)</td>
<td>0</td>
</tr>
<tr>
<td>3619</td>
<td>CustomObjectReservationSearch page</td>
<td>CustomObjectReservationSearch</td>
<td>(null)</td>
<td>0</td>
</tr>
</tbody>
</table>

In the highlighted example, the CustomObjectMaint page is authorized for the resourceAdmin role (identified with the number 2 in the figure).

For migration, the role resourceAdmin must be created in EM Console and associated with CustomObjectEditFlow permissions, which include CustomObjectEditFlow and CustomObjectSummaryFlow.

Migrating Authorization Policy for UI Actions

Every action in a UIM work area requires authorization. For example, a user must be authorized to delete a logical device. These authorization policies are defined using the EM Console.

Figure 7–3 shows a pre-7.2 SECURITYPOLICY table with an action highlighted.
In the highlighted example, a custom security policy has been defined for a new action called TRASH. Only users in the resourceAdmin role (identified by the number 2) are authorized to execute this action. In Figure 7–1, the user mary is assigned to this role.

To support this scenario in UIM 7.2, a new resource permission has to be defined using the EM Console for the application policy that maps to the user mary and the application role resourceAdmin.

The following list includes the characteristics of the resource permission:

- Permissions Class: oracle.security.jps.ResourcePermission
- Resource Name: resourceType=PAGE_ACTION, resourceName=CustomObject.TRASH
- Permissions Actions: view

**Entity-level Authorization**

In UIM 7.2, entity-level authorization is not supported by default, but you can still implement it by using extension points.

Entity-level authorization is based on the permissions that are set on an entity during create/update calls. If entity-level authorization is enabled, permissions are stored in the PERMISSIONS column of the corresponding entity table.

If you have used this feature in pre-7.2 releases, the permissions are preserved in the schema after the database upgrade. As a result, you can still implement entity-level authorization using the extension points.

In Figure 7–4, the logical device with the ID 1 has permissions set to all access to all users except joe.

See UIM System Administrator’s Guide for more information.
Several user interface changes in Oracle Communications Unified Inventory Management (UIM) 7.2 may require you to reenter some values from the previous release.

**Localization**

If you have localized UIM in a previous release, you must redo the work in UIM 7.2.

In releases prior to 7.2, UI localization was done using properties files, which are part of the Tapestry framework. In Jupiter, UI localization is done using XLF files, which are part of the ADF framework.

See *UIM Developer’s Guide* for information about localization, including examples of using XLF files.

**Customization**

If you customized the UIM user interface in a previous release, you must redo the work.

In releases prior to 7.2, UI customization was done using the Tapestry framework. In UIM 7.2, you use the ADF framework. See *UIM Developer’s Guide* for more information.

**Organizer**

In releases prior to 7.2, Search and Summary pages could be added to the Inventory Organizer. Equivalent functionality is achieved using the Favorites feature in UIM 7.2. See the UIM online Help for more information.

**Externally Referenced URL**

In UIM 7.2, you can obtain an externally referenceable link for a work area by selecting **Link to Page** from the **Help** menu. See UIM online Help for more information. If you are using direct links to UIM pages for pages in a previous release, you must replace them with new links.

**Bread Crumbs**

The bread crumbs feature in UIM has been removed for 7.2.0. This has been replaced by the new Recent Items feature that allows you to navigate quickly to your most
recent work areas. When you exit and re-enter UIM, your Recent Items are saved and available. See the UIM online Help for more information.

**My Navbar**

The ability to customize the navigation bar has been removed in 7.2.0. This feature has been replaced by the Favorites feature. See the UIM online Help for more information.

**My Desktop**

The My Desktop feature has been removed from UIM 7.2. The features it provided, such as changing the font size, are usually provided by the web browser.

**Find Links**

The Find Links feature has been removed from UIM in 7.2. Most links are readily visible and available in the Tasks panel.
Oracle Communications Unified Inventory Management (UIM) 7.2 enable you to export data via the rules mechanism in the same manner as with previous releases. Data continues to be exported in XML format. The contents and format of the control file (config.txt) used for data export has changed, however.

This chapter outlines the changes to the export control file and how you use it.

Entity Class No Longer Used

Previously the control file contained a list of fully qualified class names representing the entities to be exported as shown in the following example:

`oracle.communications.inventory.api.entity.Specification#true`

These entries should be removed from the control file.

Boolean Expression No Longer Used

Previously, class name statements included `#true`. Using this causes all specification rows to be exported from the database because the expression is always true. Remove `#true` as shown in the following example:

`oracle.communications.inventory.api.entity.Specification#name=='CISCO6400'`

Use Queries Instead of Classes and Boolean Expressions.

In UIM 7.2, you should use a JPQL query or queries in the control file to identify the date to be exported, as shown in the following example:

```sql
SELECT X FROM Specification X where X.name = 'CISCO6400'
SELECT X FROM Specification X where X.name = 'CISCO6410'
```

In this case `Specification` is not the database table name; it is the UIM entity name. That is the same as in the previous release.

RelationshipToInclude Parameter

You still use the `relationshipToInclude` parameter in the control file for UIM 7.2. The following values continue to be supported:

- **All**: Export all related entities for any entity returned by queries.
- **None**: Do not export any related entities.
UIM Meta Entities

- **Meta**: Export related entities of types defined as Meta entities by the UIM model. See "UIM Meta Entities".

- **Data**: Export related entities that are not defined as Meta entities.

You can have one and only one `relationshipToInclude` parameter in the control file.

**UIM Meta Entities**

The following entity types are defined as Meta entities in UIM model.

- Specification
- SpecificationRel
- SpecificationType
- RuleSetEntity
- InvGroupRef
- InvGroupRel
- InventoryGroup
- InventoryConfigSpecOption
- ExtensionPoint
- EnabledExtensionPoint
- ExtensionPointRuleSet
- DiscreteCharSpecValue
- RangeCharSpecValue
- CharacteristicSpecRel
- CharacteristicSpecValue
- CharacteristicSpecUsage
- CharacteristicSpecification
- CharInventoryConfigSpecOption
- EquipmentSpecificationIlogData
- Media
- SpecificationMediaRel
- UnitOfMeasure
- TypeOfMeasurement
- CapacityType

**Sample Export Control Files**

This section includes the content of several sample export control files. Note that the default value for `duplicateAction` is `update`.

**Sample 1**

```sql
SELECT X FROM Specification X
relationshipsToInclude=Meta
duplicateAction =update
```
Sample 2

SELECT X FROM Specification X where X.name=='Cisco Router 999'
SELECT X FROM Specification X where X.name=='Network 1'
relationshipsToinclude=All
duplicateAction =error

Sample 3

SELECT X FROM EquipmentSpecification X where X.vendornname=='Lucent'
relationshipsToinclude=Data