Developing and Deploying Customizations in Oracle E-Business Suite Release 12.2 (Doc ID 1577661.1)

To Bottom

This document contains the following topics:

A PDF version of this document is available here.

Part 1: Developing and Deploying Customizations in Oracle E-Business Suite Release 12.2

Section 1.1: Working with Editions

Note: This section replaces the section "Working with Editions" in Chapter 6, "Developer Guidelines for Customizations in an Online Patching-Enabled Environment," of the Oracle E-Business Suite Developer's Guide , Part No. E22961, in the Release 12.2.2 documentation library.

An Oracle E-Business Suite Release 12.2 installation now includes two editions (versions) of the application code and seed data. The file system contains two complete copies of the Oracle E-Business Suite and application-tier technology files. In the database, we use the Edition-based Redefinition feature to create a new database edition for each online patching cycle.

The "Run Edition" is the code and data used by the running application. The Run Edition includes a complete application-tier file system along with all objects and data visible in the default edition of the database. As a developer, you will connect to the Run Edition whenever you are engaged in normal development activity on the system.

The "Patch Edition" is an alternate copy of Oracle E-Business Suite code and seed data that is updated by Online Patching. The Patch Edition includes a complete copy of the application-tier file system and editioned database code objects. The Patch Edition is only usable when an Online Patching session is in progress. End users cannot access the Oracle E-Business Suite Patch Edition, but as a developer you may need to connect to the Patch Edition of a system when applying patches or debugging problems with Online Patch processing.

The Oracle E-Business Suite application-tier files are installed in a base directory of the customer's choosing. Within that base directory you will now find three important sub-directories:

- fs1 - file system 1 (either run or patch edition)
- fs2 - file system 2 (alternate of file system 1)
- fs_ne - non-editioned file system, for data files

The fs1 and fs2 directories contain the Run Edition and Patch Edition files for Oracle E-Business Suite. The "run" and "patch" file system designation will switch back and forth between fs1 and fs2 for each patching cycle. The file system designation is automatically maintained by Online Patching in the Applications Context file and the FILE_EDITION environment variable in the environment script for each file system:

```
$ # Change to EBS base directory (example)
$ cd /u01/R122_EBS

$ # Show file system designations
$ grep FILE_EDITION= */EBSapps/appl/*.env

$ # Show file system designations
$ grep FILE_EDITION= */EBSapps/appl/*.env

fs1/EBSapps/appl/exzd122x_example.env:FILE_EDITION="patch"
fs2/EBSapps/appl/exzd122x_example.env:FILE_EDITION="run"
```

In the above example, 'fs2' is the Run Edition file system, and 'fs1' is the Patch Edition.

Section 1.1.1: Connecting to the Run Edition

The Run Edition file system and database edition are used by the running application. Normal development activity (writing and testing new code) will also take place in the Run Edition of a development environment.

Oracle E-Business Suite Release 12.2.3 and later includes a script to set the run or patch edition environment by edition type. The script is called "EBSapps.env" and is found in the base directory of an Oracle E-Business Suite application-tier installation.

```
$ source /u01/R122_EBS/EBSapps.env run
E-Business Suite Environment Information
----------------------------------------
RUN File System : /u01/R122_EBS/fs2/EBSapps/appl
PATCH File System : /u01/R122_EBS/fs1/EBSapps/appl
Non-Editioned File System : /u01/R122_EBS/fs_ne
DB Host: example.com Service/SID: exzd122x

Sourcing the RUN File System ...
$ echo $FILE_EDITION
run
$ sqlplus <apps_username>

SQL> select ad_zd.get_edition_type from dual;
GET_EDITION_TYPE
----------------
RUN
```

Section 1.1.2: Connecting to the Patch Edition

The Patch Edition contains a copy of the application code that can be modified by Online Patching. A developer may need to connect to the Patch Edition of an Oracle E-Business Suite installation in order to apply patches by hand, or to investigate problems with Online Patch processing.

Warning: It is only safe to connect to the patch edition while an Online Patching session is in progress. Specifically, the Patch Edition is active after the "adop phase=prepare" operation, and persists until the cutover or abort operation is run.

Connect to the patch edition using the EBSapps.env script as follows:

```
$ source /u01/R122_EBS/EBSapps.env patch
E-Business Suite Environment Information
----------------------------------------
RUN File System : /u01/R122_EBS/fs2/EBSapps/appl
PATCH File System : /u01/R122_EBS/fs1/EBSapps/appl
Non-Editioned File System : /u01/R122_EBS/fs_ne
DB Host: example.com Service/SID: exzd122x

Sourcing the PATCH File System ...
$ echo $FILE_EDITION
patch
$ sqlplus <apps username>
SQL> select ad_zd.get_edition_type from dual;

GET_EDITION_TYPE
----------------
PATCH
```

The application-tier Patch Edition environment is configured to connect to the database patch edition by default. If a database patch edition is not active, then attempting to connect to the database from the application-tier patch edition environment will fail.

Note: When you connect to the patch edition using the command

```
$ source <INSTALL_BASE>/EBSapps.env patch
```

it sets the TWO_TASK, APPS_JDBC_URL, and AD_APPS_JDBC_URL environment variables (among many others), which define the database connection to use. When running a command line tool (especially Java programs), you must figure out which of the available environment variables values can be used to construct a command line for the tool. For example, in the case of XDOLoader, you should use AD_APPS_JDBC_URL, as in:

```
$ java oracle.apps.xdo.oa.util.XDOLoader DOWNLOAD \
-DB_USERNAME <APPS username> -DB_PASSWORD <APPS password> -JDBC_CONNECTION "$AD_APPS_JDBC_URL"
```

Section 1.1.3: Displaying Edition Status

To help keep track of what environment and edition you are connected to, it can be helpful to set the TWO_TASK or FILE_EDITION environment variable as your shell prompt.

```
$ PS1='$TWO_TASK> '
zd122_patch>
```

You can find out whether a system is in an Online Patching cycle using the "adop -status" command.

```
$ adop -status

Enter the APPS password:

Current Patching Session ID: 60

Node Name Node Type Phase Status Started Finished Elapsed
---------- --------- ----- ------ ------- -------- -------
example master PREPARE COMPLETED 02-JUL-13 04:03:25 -07:00 02-JUL-13 05:03:32 -07:00
1:00:07 APPLY COMPLETED 09-JUL-13 12:20:45 -07:00 09-JUL-13 01:23:00 -07:00
1:02:15 CUTOVER COMPLETED 10-JUL-13 09:11:41 -07:00 10-JUL-13 09:18:47 -07:00
0:07:06 CLEANUP COMPLETED 10-JUL-13 09:29:53 -07:00 10-JUL-13 09:52:50 -07:00
0:22:57
```

If the PREPARE status is COMPLETED and the CUTOVER status is not COMPLETED, then an online patching session is in progress and it is valid to connect to the patch edition of the environment. You can also see the names of past and present database editions using the ADZDSHOWED.sql script.

```
$ sqlplus <apps username> @ADZDSHOWED
"---- Editions ----"
Edition Name    Type     Status   Current?
--------------- -------- -------- --------
ORA$BASE                 RETIRED
V_20120510_1507 OLD      RETIRED
V_20120510_1547 RUN      ACTIVE   CURRENT
V_20120511_1528 PATCH    ACTIVE
```

The script lists the existing database editions and identifies the OLD, RUN, and PATCH editions. The Current flag indicates which edition you are currently in. From SQL*Plus it is possible to change your current edition.

```
SQL> exec ad_zd.set_edition('PATCH')
```

Section 1.1.4: Tools and Scripts for Edition-Based Development

The examples in this guide use various SQL*Plus scripts and command line tools like adop, xdfgen.pl and xdfcmp.pl. The scripts and tools used in Online Patching are often dependent on a specific code level in the rest of the system, so when using an Oracle E-Business Suite environment for development make sure to use the scripts and tools that come with that environment. Connect to the application-tier host for your development environment and source the Run Edition environment file.

```
$ source /u01/R122_EBS/EBSapps.env run
              ...
$ which adop
/u01/R122_EBS/fs_ne/EBSapps/appl/ad/bin/adop
$ which xdfgen.pl
/u01/R122_EBS/fs2/EBSapps/appl/fnd/12.0.0/bin/xdfgen.pl
$ which xdfcmp.pl
/u01/R122_EBS/fs2/EBSapps/appl/fnd/12.0.0/bin/xdfcmp.pl
```

There are a number of SQL*Plus scripts that can provide useful information about the state of your editioned development environment. All ADZD* scripts are found under $AD_TOP/sql. For convenience, you can add this directory to the SQLPATH environment variable so that you can refer to the scripts by simple name.

```
$ SQLPATH=$AD_TOP/sql; export SQLPATH
```

- ADZDDBCC - database compliance checker, shows violations of the database object development standards described later in this document. Warning: this script takes a long time to run.
- ADZDSHOWED - Show database editions and current edition.
- ADZDSHOWLOG - Show full diagnostic log for online patching infrastructure.
- ADZDSHOWLOGEVT - Show only event and error messages from online patching diagnostic log (a useful summary, without the detailed statement text).
- ADZDSHOWLOGERR - Show only error messages from online patching diagnostic log.
- ADZDSHOWEV *TABLE_SYNONYM_NAME* - Show editioning view column mapping for table.
- ADZDSHOWTAB *TABLE_SYNONYM_NAME* - Show table information and related objects.
- ADZDSHOWMV *MVIEW_NAME* - Show materialized view information and related objects.
- ADZDSHOWTS - Show important tablespace status. Ensure that you have enough SYSTEM tablespace.
- ADZDCMPED - Compare Patch Edition with Run Edition. Warning: this script may take a long time to run.
- ADZDSHOWDDLS - Show stored DDL summary by phase.
- ADZDALLDDLS - Show stored DDL statement text.
- ADZDDDLERROR - Show stored DDL processing errors and messages.
- adutlrcmp - Recompile all objects, with before/after status report. Warning: this script may take a long time to run.
- ADZDSHOWOPERR - Show online patching errors affecting the run edition or table data in the patch edition. (Delivered in R12.AD.C.Delta.13).
- ADZDSHOWTRIG - Show trigger information, including crossedition trigger column reference detail. (Delivered in R12.AD.C.Delta.13).

The following scripts are for experts:

- ADZDSHOWOBJS - Show Object Summary per edition. Counts of actual and stub (inherited) editioned object per edition.
- ADZDSHOWAOBJS - Show Actual Objects in the current edition. These are the editioned objects that have been changed by the patch.
- ADZDSHOWIOBJS - Show Inherited Objects in the current edition. These are the editioned objects that remain untouched in the patch edition. This script is used to confirm that the adop actualize_all command has worked properly.
- ADZDSHOWCOBJS - Show Covered Object Summary per edition. Count of objects in old editions that have a replacement in the run edition. This script is used to confirm that the adop cleanup command has worked properly.
- ADZDSHOWCOBJX - Show Covered Object List. List of objects in old editions that have a replacement in the run edition.
- ADZDSHOWSM - Show Seed Manager status. ADZDSHOWTM - Show Table Manager status. ADZDSHOWAD - AD (online patching) database object status
- ADZDSHOWSES - Show sessions connected to the database (by edition).
- ADZDSHOWDEP OBJECT_NAME - Show objects that OBJECT_NAME depends on.
- ADZDSHOWDEPTREE OBJECT_NAME - Show full dependency tree of objects that OBJECT_NAME depends on.

Section 1.2: Applying Online Patches

Note: This section should follow the section "Working with Editions" in Chapter 6, "Developer Guidelines for Customizations in an Online Patching-Enabled Environment," of the Oracle E-Business Suite Developer's Guide , Part No. E22961.

Before developing on an editioned application system, you should understand how online patches are applied to that system. Application development is done on the Run Edition of a development system, while an online patch is always applied to the Patch Edition of a target system. The online patch may take the form of a Manual Patch or an Oracle patch.

- A manual patch consists of a set of files plus a set of installation actions that are run to apply the changes to a target system. The procedure for applying a manual patch to an editioned system is similar to that of earlier non-editioned releases, with two important differences:
  - Manual patching actions must be run in the Patch Edition of the target system.
  - Manual patching actions that affect the file system must be repeated or copied to the alternate file system on the next patching cycle.
- An Oracle Patch consists of a set of files that may be annotated with "dbdrv" comments, which are processed by Oracle to produce a patch bundle. The patch bundle can then be applied automatically using the "adop phase=apply" command. This is the equivalent of running "adpatch" on non-editioned system. The "adop phase=apply" command runs all patch actions required to apply the update to the patch edition of the target system, and automatically handles file system synchronization on the next patching cycle.

Section 1.2.1: The Online Patching Cycle

Patches to an editioned system are normally applied within the context of an Online Patching Cycle. The online patching cycle has several phases which proceed in order.

- Prepare - creates the patch edition.
- Apply - apply Oracle or manual patches to the patch edition.
- Finalize - perform any actions required to prepare for cutover.
- Cutover - Promote Patch Edition to be the new Run Edition.
- Cleanup - remove obsolete code and data from old editions.

Online Patching Cycle phases are run using the new "adop" command line tool. Syntax for each of the phases is described below. At any time you can get adop command line help by running "adop -help". You can check the status of the patching cycle by running "adop -status".

The following sections describe how to progress through each phase. For more information on the Online Patching Cycle, refer to the *Oracle E-Business Suite Maintenance Guide*, Part No. E22954.

Section 1.2.2: Prepare

Before applying a patch, you must start an Online Patching Cycle. This is done using the adop "prepare" command. Connect to the primary application-tier node of your target system and source the run edition environment. Then run the prepare command.

```
$ source /u01/R122_EBS/EBSapps.env run
...
$ adop phase=prepare
```

The adop utility may first run the cleanup phase from the previous cycle if needed, and will then proceed to prepare the patch edition for a new Online Patching Cycle. To prepare the patch edition, adop will:

1. Validate that the system is ready to start a new patching cycle.
2. Create a new database patch edition
3. Synchronize the file system patch edition with the run edition
4. Configure the patch edition for use by the patching tools

File system synchronization may be done by applying the delta (changes) from the previous patching cycle, or by re-creating the entire patch edition file system as a fresh copy of the run edition (called "fs_clone"). When complete, check the exiting status code (success is '0'):

```
adop exiting with status = 0 (Success)
```

If there were any problems with the prepare phase, check Section 1.7: Troubleshooting and resolve the problem. Then run the prepare command again.

After a successful prepare phase, the database and file system patch edition will contain a copy of the run edition code and seed data. You can now apply Oracle patches and manual patches to the patch edition.

Section 1.2.3: Apply

Once the Patch Edition is prepared, you can apply any number of Oracle patches or manual patches to the patch edition. Changes to the patch edition are isolated from the run edition, which is still available for use.

Apply an Oracle Patch

Before applying an Oracle patch, you must first download the patch bundle from Oracle through the web user interface (support.oracle.com). The downloads will be in the form of ZIP files. Place the ZIP files in the "$PATCH_TOP" directory on the application-tier installation of your target application system, and then unzip all ZIP files.

Oracle patches are applied to the patch edition using the "adop phase=apply" command. The command accepts a "patches=..." parameter where you can specify a single patch or a comma-separated list of patches.

```
$ adop phase=apply patches=16605855
...
$ adop phase=apply patches=15111111,15222222
...
```

Note that the adop command will apply patches to the patch edition no matter what edition your current environment is set to.

If the adop apply commands fail, check Section 1.7: Troubleshooting and correct the problem, then run the adop apply command again, adding the "restart=yes" option.

```
$ adop phase=apply patches=16605855 restart=yes
...
```

In some cases, an error when applying a patch can be corrected by applying a replacement patch. You can abandon an existing failed patch and apply the replacement patch by running the apply command with the "abandon=yes" parameter.

```
$ adop phase=apply patches=16699999 abandon=yes
...
```

Apply a Manual Patch

Manual patches must be applied to the patch edition of a target system "by hand". Do this by changing to the patch edition environment and manually running the patching actions necessary to install the update. The manual patch actions are identical to those you would take when applying manual patches to a non-editioned system; the only difference is that on an editioned system, these actions take place in the patch edition.

Manual patching actions normally involve the following steps:

Copy patch files to their destination directories in the patch edition.

Run any commands necessary to deploy changes to the file system.

Run any commands necessary to deploy changes to the database.

Run any commands necessary to generate or compile dependent files or objects.

Update the custom synchronization driver to include any file system actions that must be run again on the next prepare phase, in order to synchronize the alternate file system. See Section 1.5.4: Adding Entries to the Custom Synchronization Driver File.

The exact commands needed to apply a manual patch vary by the type of files or database objects being patched. These required deployment commands for each file and object type are discussed later in this document.

The following is a simple example of installing a new server PL/SQL package.

```
$ source /u01/R122_EBS/EBSapps.env patch
...

$ cd $PATCH_TOP/manual_000
$ apply_fs.sh

# apply patch to file system
cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql

$ apply_db.sh

# apply patch to database
sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZUTILS.pls
sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZUTILB.pls
```

After applying an Oracle patch or a manual patch you can look at the patch edition file system or database status to verify that the patching actions were successful and that the resulting patch edition code and seed data are as expected. When you have sourced the patch edition environment, the default database connection goes to the patch edition. Although you cannot run the application user interface or program code in the patch edition, it is possible to connect to the database via SQL*Plus or other tools and confirm that the desired changes have been successfully implemented. To confirm the updates of the previous manual patching example, you could do the following:

```
$ source /u01/R122_EBS/EBSapps.env patch
...

$ sqlplus <apps username>
Enter password: password
SQL> show errors package XYZ_UTIL
SQL> show errors package body XYZ_UTIL
SQL> quit
```

Once all patching actions are complete and validated, you may proceed to the finalize phase.

Section 1.2.4: Finalize

The finalize phase is used by the Online Patching tool to perform any final actions needed to make the system ready for the fastest possible cutover. The finalize command is run as follows:

```
 $ adop phase=finalize
```

If the finalize command returns an error, the system is not ready for cutover. In this case, check Section 1.7: Troubleshooting, correct the problem and run the finalize command again.

After successful completion of the finalize phase, the system is ready for cutover, but you do not need to run the cutover right away; you can delay running cutover until a convenient or predetermined time in the future. You may also apply additional patches if needed, but you will need to run the finalize phase again after doing so.

Section 1.2.5: Cutover

The cutover phase will configure the patch edition to become the new run edition, and restart the application on this new run edition.

```
 $ adop phase=cutover
...
$ source /u01/R122_EBS/EBSapps.env run
```

After successful completion of the cutover phase, the application will be up and running on the new edition, ready for use. Since the run/patch designation of the dual file systems are swapped during cutover, you must re-source the run edition environment directly after cutover.

Important: Remember to re-source the run edition environment directly after cutover.

Section 1.2.6: Cleanup

The cleanup phase will remove unnecessary code and data from old editions that are no longer needed by the application. Cleanup should be run after cutover, at any time before the next prepare phase. It is best to run cleanup immediately after cutover so that there is no delay when preparing the next online patching cycle. There are three levels of cleanup available:

- quick - the minimal cleanup required before starting the next patching cycle.
- standard - (default) removes obsolete code and seed data from old editions.
- full - removes all obsolete code, data, and table columns.

Standard cleanup is the default, and is recommended to avoid a buildup of obsolete code objects in old editions.

```
 $ adop phase=cleanup
```

Use quick cleanup when you need to start the next patching cycle as soon as possible.

```
$ adop phase=cleanup cleanup_mode=quick
```

Use full cleanup to remove obsolete table columns. Full cleanup is required after aborting an online patching cycle.

```
$ adop phase=cleanup cleanup_mode=full
```

Section 1.2.7: Special Patching Actions

For completeness, the following actions are also listed here. Refer to the *Oracle E-Business Suite Maintenance Guide*, Part No. E22954 for more information on these commands.

FS Clone

The "fs_clone" command recreates the patch edition file system by making a full copy of the run edition file system. This command is required when the patch edition file system cannot be automatically synchronized with the run edition file system by adop. Run the fs_clone command if you have done any of the following actions:

- Applied middle-tier technology patches to the run edition
- Changed run edition files manually (outside of online patching)
- Aborted an online patching cycle

The fs_clone command is run as follows:

```
$ adop phase=fs_clone
```

Abort

If an online patching cycle has failed for some reason that cannot be corrected, you can abort the online patching cycle and return to normal operation. The adop "abort" command drops the database patch edition and removes or abandons any changes made during the the online patching cycle. If the online patching cycle has completed the cutover phase, then you can no longer abort the patching cycle.

When aborting an online patching cycle, you must also run the cleanup and fs_clone commands to fully eliminate changes from the patching cycle. The abort command is run as follows:

```
$ adop phase=abort
$ adop phase=cleanup
$ adop phase=fs_clone
```

Section 1.3: Developing Customizations

For Oracle E-Business Suite developers, development activity can be considered in two parts:

1. Application Development: Creating or revising part of an application definition, functional testing, source control of updates.
2. Patch Development: Creating a patch to deliver and install the changed application definition on a target system, patch testing.

Application Development is done in the Run Edition of an Oracle E-Business Suite development environment. The runtime application user interface and other services can only operate against the run edition of a system. During application development, the developer is directly modifying the definition of the running application in a development database. Once satisfied with the updated application definition, the developer proceeds to patch development.

Patch Development involves collecting or extracting the files that contain the application definition to be patched, along with any other logic necessary to deliver, build and deploy these changes to the target system. Customers may create manual patches.

To create a manual patch, the developer creates a patch directory containing the application definition files to be delivered, along with an "apply" script. The apply script contains the commands to deliver, build, and deploy the application definition to the target system. Patch apply scripts can be initially tested in the run edition of a development database, but final testing must take place in the patch edition of a test system during an online patching cycle.

The rest of this section describes how to create and patch the various types of application definition changes in an editioned application environment.

Application-Tier Files

Application-tier files (also known as middle-tier files) are those files that are only used by middle-tier application services such as the OA Framework, Forms, Reports, Concurrent Programs, JSP files, and so on. An application developer working on a development system can create or modify middle-tier files directly in the run edition file system. Changes to some file types require a build or deployment action to be performed before the change will take effect. Any required build or deployment actions will be explained in the specific instructions for each file type below, but the general development approach for middle-tier file changes is as follows:

1. Connect to the run edition of your development environment.
2. Check out the desired source files from the source control system.
3. Edit source files to implement desired update.
4. Build/deploy changes into the run edition of the development environment and test.
5. When development is complete, check in the changes and make note of new files and versions.
6. Create a patch for the changed files. For manual patches, create a script that copies changed files to the target system and then run any build or deploy actions that are required.

Section 1.3.1: Setting Up an Environment for Customizations

If you are developing customizations for the first time, begin by setting up your custom application on your development environment. See: Overview of Setting Up Your Application Framework, *Oracle E-Business Suite Developer's Guide*.

As part of setting up your application, use the AD Splicer utility (adsplice) to register your custom application as a product within Oracle E-Business Suite. For instructions on running adsplice, see: Applications DBA System Configuration Tools, *Oracle E-Business Suite Setup Guide*, and Applications DBA System Maintenance Tasks and Tools, *Oracle E-Business Suite Maintenance Guide*.

Note: In Release 12.2, you should use adsplice to register your application in order to ensure that the application is set up for online patching. Do not use the Applications window to register applications in this release.

Note: When installing or upgrading to Release 12.2, do not run adsplice until you have applied the 12.2.2 Release Update Pack. Running adsplice before your instance is at the 12.2.2 code level may cause file synchronization issues.

You can use Patch 3636980, "Support Diagnostics (IZU) patch for AD Splice", to help you create your custom application. See: *Creating a Custom Application in Oracle E-Business Suite Release 12.2*, My Oracle Support Knowledge Document 1577707.1.

On your development environment, you should invoke adsplice from the run file system. Connect to the run file system as described in Section 1.1.1: Connecting to the Run Edition. Then run the `adsplice` command.

In Oracle E-Business Suite Release 12.2, adsplice performs the following steps:

- Makes the new user edition-enabled.
- Enables Edition-Based Redefinition (EBR) for the custom objects.

When you start the next online patching cycle, the prepare phase will run fs_clone to synchronize the two file systems.

Note: If you upgraded your environment from an earlier release to Release 12.2, then you should run adsplice for your custom application again after the upgrade, using the same application ID and application name as when you originally added your custom application. Running the Release 12.2 version of adsplice after the upgrade helps ensure that the custom top folder for your application will be included when the two file systems are synchronized during online patching.

If your customizations will include custom Java or BC4J code or extensions, apply the following patches to your development environment in hotpatch mode using the AD Online Patching utility (adop). For instructions on running adop, see: The adop Utility, *Oracle E-Business Suite Maintenance Guide*.

- 17217965:R12.TXK.C (TEMPLATE CHANGE REQUIRED TO UPLOAD THE CLASS FILES RELATED TO CUSTOMIZATIONS)
- 17217772:R12.AD.C (NEED UTILITY TO GENERATE CUSTOMALL.JAR)

Section 1.3.2: Building Customizations

After setting up your development environment, build your customizations according to the developer's guide for the product or component you are customizing, as well as any guidelines in Section 1.6: Component-Specific Steps for Application Tier Objects.

For customizations developed directly in the Oracle E-Business Suite instance, you should download the custom object files that you will deploy to your production environment.

1. Connect to the run edition file system on your development environment.
2. Use the appropriate utility for your product or component to download the custom object files.

For customizations developed in a tool outside the Oracle E-Business Suite instance, you should save the custom object files from that tool. To deploy the custom object files in your development environment for testing, perform the following steps:

1. Connect to the run edition file system on your development environment.
2. Copy the custom files to the appropriate directory on the run edition file system.
3. If you copied any custom files under the $JAVA_TOP directory, run the adcgnjar utility to generate and sign a JAR file containing these files. When prompted, enter the user name and password of the APPS user. See Section 1.5.3: Running the adcgnjar Utility.
4. If necessary, use the appropriate utility for your product or component to upload the custom files to the database.
5. Add entries for the custom files to the custom synchronization driver file to ensure that the adop utility synchronizes these files between the run file system and the patch file system the next time you run the prepare phase. See Section 1.5.4: Adding Entries to the Custom Synchronization Driver File.

Section 1.4: Developing and Deploying Custom Code and Custom Database Objects

For more information on database object development standards, see: Part 2: Database Object Development Standards for Online Patching.

Section 1.4.1: Recommended Approaches for Deploying Custom Code

When patching custom code into an Oracle E-Business Suite installation, there are two recommended approaches.

1. Apply customizations as an online patch: The custom code is patched into the patch edition when an online patching cycle is open. The patch must follow Online Patching Development Standards. In this case, the custom patch acts like any other Oracle E-Business Suite online patch and receives the same benefits (no additional downtime, ability to review patching results before cutover, ability to abort the patch if there is trouble). This is the most recommended approach but requires compliance to the full set development standards.
2. Apply customizations as a downtime patch: The custom code is patched into the run edition, either during an extended online patching cutover downtime, or as a separate downtime event. In this case, the custom patch need only comply with only a minimal set of development standards, and in particular, the patch is free to use traditional table upgrade scripts as the method for data upgrade. Downtime patching of customizations is recommended if the customer is not sensitive to downtime and does not want to ensure that their custom development meets full online patching development standards.

An approach that explicitly discouraged is hot patching of customizations. Applying custom changes to the run edition of the system while the system is online carries the risk of causing large chain-reaction invalidation of objects in the dependency tree of whatever is getting patched. This can cause runtime failure in user sessions, and the code may temporarily be in an inconsistent state which can lead to data corruption, runtime exceptions, or other unexpected behavior while the patch is being applied.

Note: As a customer, you do not need to commit to only the online approach or only the downtime approach; that is, you can make this decision on a case-by-case basis. Online patching of code-only changes requires no extra effort from the developer compared to downtime patching, because the process of actually applying code changes is identical (once you have set your environment to the correct edition). So you can do simpler patches online and reserve downtime patching for situations where the patch will modify tables or other non-editioned objects in a way that is not compliant with online patching.

Section 1.4.2: Editioned Database Objects

Note: This section replaces the section "Editioned Database Objects" in Chapter 6, "Developer Guidelines for Customizations in an Online Patching-Enabled Environment," of the Oracle E-Business Suite Developer's Guide, Part No. E22961, in the Release 12.2.2 documentation library.

Editioned Database Objects may have a different definition in each database edition. This means that such objects can be created or replaced in the patch edition without affecting the running application. Editioned database object types are:

- View (Ordinary)
- PL/SQL Package
- PL/SQL Trigger
- User-Defined Type (Editioned)
- Synonym
- Virtual Private Database Policy

For more information on these objects, refer to the *Oracle Database Administrator's Guide.*

Step 1: Create or Replace Editioned Database Objects in your development database:

An application developer can create or replace editioned database objects in the run edition of a development database using whatever scripts or tools they normally use. Typically this involves editing SQL scripts that contain DDL statements, and then applying the scripts to the development database. For example:

```
sqlplus <apps_username> @XYZUTILS.pls"
sqlplus <apps_username> @XYZUTILB.pls"
sqlplus <apps_username>
                exec ad_zd.compile
                quit
```

If your application changes will cause significant object invalidation in the development database, you may wish to call the "ad_zd.compile" procedure to recompile invalid objects in the run edition.

Note: Developers often use the user_objects or all_objects data dictionary view to confirm that there are no unexpected invalid objects. Due to a database limitation these dictionary views only return correct object status after running a full compilation procedure (utl_recomp.recomp_parallel or ad_zd.compile). As a workaround, you can check object status using the ad_objects view included with online patching.

```
select * from ad_objects where status='INVALID';
```

After deploying changes, confirm that there are no unexpected invalid objects and then test your changes in the running application. When satisfied, make note of the changed DDL scripts and proceed to the next step.

Step 2: Create the patch

Patch files in the above example would be:

- fnd/patch/115/sql/XYZUTILS.pls
- fnd/patch/115/sql/XYZUTILB.pls

The manual apply actions for the file system would be:

```
cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
```

The manual apply actions for the database would be:

```
sqlplus <apps_username> @$FND_TOP/patch/115/sql/XYZUTILS.pls
sqlplus <apps_username> @$FND_TOP/patch/115/sql/XYZUTILB.pls
```

Section 1.4.3: Effectively-Editioned Database Objects

Note: This section replaces the sections "Tables" and "Materialized Views" in Chapter 6, "Developer Guidelines for Customizations in an Online Patching-Enabled Environment", of the Oracle E-Business Suite Developer's Guide, Part No. E22961, in the Release 12.2.2 documentation library.

Section 1.4.3.1: Tables

Since the application is still running during an online patch (and the application data is continuously changing), it is not possible to upgrade application data using a one-time update script. Instead we will need to use a new technique involving Editioning Views and Crossedition Triggers, described below.

Note: This section describes how to develop and patch ordinary application data tables. But there are some special types of tables that have additional or alternate standards and procedures. If you are working with one of these special table types, please consult that section of the guide instead.

Create a new table

This example will show how to develop and patch a new table on an editioned development environment. Suppose we want to create a table that holds "service information" per user account for some application with the following logical table structure:

```
XYZ_USER_SERVICE

Name                                      Null?    Type
----------------------------------------- -------- --------------
USER_ID                                   NOT NULL NUMBER
  -- PK, FK to FND_USER.USER_ID
SERVICE_TYPE                              NOT NULL VARCHAR2(8)
  -- 'BASIC'    - normal service
  -- 'PREMIUM'  - premium service
COMMENTS                                           VARCHAR2(1000)
```

1. Create the initial table definition in your development database.

   In this example we use SQL*Plus to run the required DDL. This step includes creation of any required indexes, storage properties, and so on. As with all development, you should be connected to the Run Edition of your EBS development environment.

   ```
    create table APPLSYS.XYZ_USER_SERVICE
      (
             USER_ID
             NUMBER(15)     not null,
             SERVICE_TYPE    VARCHAR2(8)    not null,
             COMMENTS        VARCHAR2(1000)
            )
            tablespace APPS_TS_TX_DATA
           /
           create unique index APPLSYS.XYZ_USER_SERVICE_U1
            on APPLSYS.XYZ_USER_SERVICE ( USER_ID )
            tablespace APPS_TS_TX_IDX
    /
   ```

   Please avoid using official database constraints for Primary Key and Unique Key enforcement. Unique indexes achieve the goal and are easier to manage under Online Patching.

2. Upgrade the table for Online Patching using the AD_ZD_TABLE.UPGRADE procedure.

   This will generate an Editioning View (EV) for the table and then create an APPS synonym that points to the Editioning View.

   ```
   exec ad_zd_table.upgrade('APPLSYS', 'XYZ_USER_SERVICE')
   ```

The table is now ready for use from the APPS schema. The generated EV is named XYZ_USER_SERVICE# and looks exactly like the table at this point. When the table structure is patched in the future, the EV will serve to map logical column names (used by the application code) to the table columns that store the data in each edition. You can see a display of the EV column mapping with the ADZDSHOWEV.sql script:

```
$AD_TOP/sql/ADZDSHOWEV.sql XYZ_USER_SERVICE
-- EV Column Mapping
VIEW_COLUMN                    ->   TABLE_COLUMN
------------------------------ ---- --------------------
USER_ID                        =    USER_ID
SERVICE_TYPE                   =    SERVICE_TYPE
COMMENTS                       =    COMMENTS
```

Now we can add some data to the table for demonstration purposes:

```
insert into xyz_user_service (user_id, service_type, comments)
       values (0, 'PREMIUM', 'Big Spender');
insert into xyz_user_service (user_id, service_type, comments)
       values (2, 'BASIC', 'Mr Prudent');
commit;
```

3. Extract the table definition from your development database using the xdfgen.pl utility.

   Due to a database requirement you must first insert at least one row into the table before extraction will work.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user> XYZ_USER_SERVICE
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password> XYZ_USER_SERVICE
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID> XYZ_USER_SERVICE
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_USER_SERVICE
   ```

   This produces a file called 'xyz_user_service.xdf' that contains the definition of the table along with any related indexes, sequences, and policies.

4. Create the patch.

   Patch Files:

   fnd/patch/115/xdf/xyz_user_service.xdf version 1

   Note that version 1 of the file xyz_user_service.xdf contains the initial table definition. In the example in the next section, you create a new patch that introduces a change to the table definition which is shipped in the version 2 of the file xyz_user_service.xdf. In subsequent sections you create new patches that introduce changes that are shipped in later versions of the file (version 3, version 4, and so on).

   Manual apply phase actions for the file system:

   ```
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual apply phase actions for the database:

   For instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_service.xdf <apps_user>
   ```

   For instances on R12.AD.C.Delta.7 and earlier:

   ```
   xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_user_service.xdf <apps_user>/<apps_password>
   ```

   When the patch is applied, XDF will create the table and index, and will automatically call the AD_ZD_TABLE.UPGRADE procedure to generate the editioning view and APPS table synonym.

Add a new column to a table

This step demonstrates adding a new logical column to a table (as opposed to revising an existing logical column, which we will cover in a later section). To demonstrate this procedure, will add a new flag to our example service table that indicates whether the service is enabled. The desired logical table structure is as follows:

```
XYZ_USER_SERVICE
Name                                      Null?    Type
----------------------------------------- -------- --------------
USER_ID                                   NOT NULL NUMBER
   -- PK, FK to FND_USER.USER_ID
SERVICE_TYPE                              NOT NULL VARCHAR2(8)
   -- 'BASIC'    - normal service
   -- 'PREMIUM'  - premium service
COMMENTS                                           VARCHAR2(1000)
 ==>  SERVICE_STATUS                      NOT NULL VARCHAR2(8)
 ==>   -- 'ENABLED'  - service is active
 ==>   -- 'DISABLED' - service is not active.
```

1. Create the new column in your development database.

   We can do this in SQL*Plus as follows:

   ```
    alter table APPLSYS.XYZ_USER_SERVICE
                add (SERVICE_STATUS varchar2(8) default 'ENABLED' not null)
    /
   ```

   Note: When adding a NOT NULL column, it is recommended to choose a default value. Even if the column value will be populated through application logic you should still specify a default value for a NOT NULL column. The default value will allow XDF/ODF to create the column with the NOT NULL constraint in a single pass. Populating a new or revised column during online patching is done using a crossedition trigger which will be explained later.

2. Regenerate the editioning view using AD_ZD_TABLE.PATCH. Whenever you directly alter the structure of a table, you must call the AD_ZD_TABLE.PATCH procedure. The PATCH procedure looks at the physical table columns and then generates the editioning view which presents the logical columns for that table. The PATCH procedure is called automatically when applying table structure changes using XDF or ODF.

   ```
       exec ad_zd_table.patch('APPLSYS',
                   'XYZ_USER_SERVICE')

                    @ADZDSHOWEV XYZ_USER_SERVICE
                    -- EV Column Mapping

                    VIEW_COLUMN                    ->   TABLE_COLUMN
                    ------------------------------ ---- ------------------------------
                    USER_ID                        =    USER_ID
                    SERVICE_TYPE                   =    SERVICE_TYPE
                    COMMENTS                       =    COMMENTS
                    SERVICE_STATUS                 =    SERVICE_STATUS
   ```

   The new column is now present in the Editioning View.

3. Extract the updated table definition from your development database.

   For instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user>  XYZ_USER_SERVICE
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>  XYZ_USER_SERVICE
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_USER_SERVICE
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier::

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_USER_SERVICE
   ```

4. Create the patch.

   Patch Files:

   fnd/patch/115/xdf/xyz_user_service.xdf version 2

   Manual apply phase actions for the file system:

   ```
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual apply phase actions for the database:

   For instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
    xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_service.xdf
   ```

   For instances on R12.AD.C.Delta.7 and earlier:

   ```
    xdfcmp.pl <applsys_user>/<apps_password> $FND_TOP/patch/115/xdf/xyz_user_service.xdf
   ```

5. Test the patch.

   When the patch is applied, XDF will add the new column and automatically call the AD_ZD_TABLE.PATCH procedure on the target system.

Add a new index

This section demonstrates how to add a new index to an existing table. In the following example, we add a non-unique index on the SERVICE_TYPE attribute of our example table. The logical table structure is unchanged.

1. Create the new index in your development database.

```
create index APPLSYS.XYZ_USER_SERVICE_N1 on APPLSYS.XYZ_USER_SERVICE
       ( SERVICE_TYPE )
       tablespace APPS_TS_TX_IDX
/
```

   When adding an index it is not necessary to call the AD_ZD_TABLE.PATCH procedure, as the table structure has not changed.

2. Extract the updated table definition from your development database.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
perl xdfgen.pl <apps_user> XYZ_USER_SERVICE
```

   For all instances on R12.AD.C.Delta.7:

```
perl xdfgen.pl <apps_user>/<apps_password> XYZ_USER_SERVICE
```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_USER_SERVICE
```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_USER_SERVICE
```

   When extracting a table definition, XDF also extracts any related index definitions.

3. Create the patch.

   Patch Files:

   fnd/patch/115/xdf/xyz_user_service.xdf version 3

   Manual apply phase actions for the file system:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
```

   Manual apply actions for the database:

   For instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_service.xdf
```

   For instances on R12.AD.C.Delta.7 and earlier:

```
xdfcmp.pl <applsys_user>/<apps_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_user_service.xdf
```

4. Test the patch.

   When XDF applies the table definition, it will detect that the target database is missing the new index, and it will create the new index. Note that when the XDF is applied in the Patch Edition of a target system, the new index is initially created with an alternate name, which will then be updated to the correct index name during cutover.

Update an existing column

This section shows how to update an existing logical column. You might need to update an existing logical column either to

- Change the column definition (data type, data length, not null constraint)
- Change the column data (how data is stored in the column)

To update existing data without disturbing the running application we must create a new physical column (called a revised column) to hold the updated data. Note that online patching tools will not allow you to alter the existing physical column definition in any way, even if the change seems "compatible" with the existing column data. To make any kind of change to an existing column you must code a revised column using the procedure described in this section.

In this example, we upgrade SERVICE_TYPE codes from the original two-value scheme ('BASIC', 'PREMIUM') to a three-value scheme ('BRONZE', 'SILVER', 'GOLD'). Since the new values are not compatible with the existing application, we must use a revised physical column to hold the new data. The logical name of the column (as exposed through the editioning view) remains the same. The desired logical table structure is as follows:

```
    XYZ_USER_SERVICE
    Name                                      Null?    Type
    ----------------------------------------- -------- ---------------
    USER_ID                                   NOT NULL NUMBER
       -- PK, FK to FND_USER.USER_ID
    SERVICE_TYPE                              NOT NULL VARCHAR2(8)
          ==>     -- 'BRONZE' - cheap service (was 'BASIC')
          ==>     -- 'SILVER' - new mid-level service
          ==>     -- 'GOLD'   - best service (was 'PREMIUM')
    COMMENTS                                           VARCHAR2(1000)
    SERVICE_STATUS                            NOT NULL VARCHAR2(8)
       -- 'ENABLED'  - service is active
       -- 'DISABLED' - service is not active.
```

1. Create a revised column in your development database.

   Revised columns use a naming standard of COLUMN_NAME#REVISION, where a later REVISION tag must be alphabetically greater than the earlier revision. Since this is the first revision of the column, start with revision '1'. The data upgrade logic will be placed in a Forward Crossedition Trigger described later. Alter the table in your development database to add the new revised column, and remember to call the AD_ZD_TABLE.PATCH procedure whenever you change the table structure manually:

```
alter table APPLSYS.XYZ_USER_SERVICE
              add (SERVICE_TYPE#1 varchar2(8) default '*NULL*' not null)
/
exec ad_zd_table.patch('APPLSYS', 'XYZ_USER_SERVICE')
```

   Since the revised column is not null, specify a default value so that the column can be created with the not null constraint in a single operation. The actual value of the column will be populated by a crossedition trigger, so the default value does not matter, but it is useful to specify a default value which clearly indicates that the column is not yet populated.

```
@ADZDSHOWEV XYZ_USER_SERVICE

               -- EV Column Mapping
               VIEW_COLUMN                    ->   TABLE_COLUMN
               ------------------------------ ---- ------------------------------
               USER_ID                        =    USER_ID
               SERVICE_TYPE                   ===> SERVICE_TYPE#1
               COMMENTS                       =    COMMENTS
               SERVICE_STATUS                 =    SERVICE_STATUS
```

   Notice that after running the PATCH procedure the SERVICE_TYPE column in the EV (the logical column) is now mapped to the revised physical column. Also notice that this new column is not yet populated with data. That comes next.

2. Create a Forward Crossedition Trigger to populate the revised column.

   A Forward Crossedition Trigger (FCET) is a table trigger with a special rule about how it fires: During online patching, the FCET is created in the Patch Edition, but (being a crossedition trigger) it will only fire on changes made in the parent (Run) edition. The upgrade logic is implemented as a trigger instead of a simple update statement so that the upgrade logic can be run again on rows that are inserted or changed by the running application.

   Although the FCET is intended to be installed in the Patch Edition during an online patch, you can create and test an FCET in the Run Edition of a development database. To create an FCET, start with the Forward Crossedition Trigger Template and add the data upgrade logic to the trigger body.

   The Forward Cross-edition Trigger Template is as follows:

```
REM ---- Create FCET ----
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:     none none none sqlplus &phase=ccet \
REM dbdrv:     checkfile:~PROD:~PATH:~FILE &un_<table_owner_app_short_name>
REM ---- Apply FCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:     none none none sqlplus &phase=acet \
REM dbdrv:     checkfile:~PROD:~PATH:~FILE <table_name>_F<change_number>

REM Copyright (c) 2013 Oracle, All Rights Reserved
REM $Header$       REM <table_name>_X<change_number>.sql
REM <description of change>

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace trigger <table_name>_F<change_number>
        before insert or update on &1..<table_name>
        for each row  forward crossedition
        /* follows <previous_fcet> */ disable
begin
        <upgrade logic>
end;
/

commit;
exit;
```

For our example, the FCET looks like the following:

```
REM ---- Create FCET ----
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:     none none none sqlplus &phase=ccet \
REM dbdrv:     checkfile:~PROD:~PATH:~FILE &un_fnd
REM ---- Apply FCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet XYZ_USER_SERVICE_F1

REM Copyright (c) 2013 Oracle Corporation, All Rights Reserved
REM $Header$
REM XYZ_USER_SERVICE_X1.sql
REM     Update XYZ_USER_SERVICE SERVICE_TYPE to BRONZE/SILVER/GOLD

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace trigger XYZ_USER_SERVICE_F1
        before insert or update on &1..XYZ_USER_SERVICE
        for each row forward crossedition
        disable
begin
        if :new.service_type = 'BASIC' then
                :new.service_type#1 := 'BRONZE';
        elsif :new.service_type = 'PREMIUM' then
                :new.service_type#1 := 'GOLD';
        end if;
end;
/

commit;
exit;
```

Create the trigger with the following naming standards:

- Crossedition Trigger Script Name: <table_name>_X<change_number>.sql
  - <change_number> is incremented for each successive patch to the table
  - Example: XYZ_USER_SERVICE_X1.sql, XYZ_USER_SERVICE_X2.sql, ...
- Forward Crossedition Trigger Name: <table_name>_F<change_number>
- Reverse Crossedition Trigger Name: <table_name>_R<change_number>

For custom (manual) patches, you use the script template and remove or ignore the "dbdrv" comments. Your database apply script will include commands to install and apply the FCET.

To unit test your crossedition trigger logic you can install and apply the trigger manually in the run edition of your development database. Run the SQL script to create the trigger and then call the AD_ZD_TABLE_APPLY script to apply the trigger.

```
sqlplus <apps_user> @XYZ_USER_SERVICE_X1 APPLSYS
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY.sql XYZ_USER_SERVICE_F1
```

At this point the new column is populated. The final step of updating an existing logical column is to maintain any managed objects that may be referencing the original (now out-of-date) column. There may be indexes or materialized views that reference the physical table columns. If these objects reference obsolete table columns, they need to be updated to refer to the latest revised columns. This step can be done automatically by Online Patching.

To fix managed objects after revising an existing logical column, call the AD_ZD.FINALIZE, AD_ZD.CUTOVER and AD_ZD.CLEANUP procedures manually in your development database. These operations are normally done as part of the Online Patching Cycle, but since your development environment is not actually in an Online Patching Cycle, you must call the procedures manually.

```
sqlplus <apps_user>
                exec ad_zd.finalize
                exec ad_zd.cutover
                exec ad_zd.cleanup
                quit
```

The FINALIZE procedure creates a revised version of the index on SERVICE_TYPE. Since we are now storing the service type information in the SERVICE_TYPE#1 column, the existing index must be updated to use the new column. FINALIZE creates the revised index under an alternate name, which will be changed to the original name during the cutover phase.

The CUTOVER procedure removes the "NOT NULL" constraint on the old SERVICE_TYPE column, drops the old index, and renames the revised index to the original name. In a real Online Patch, the CUTOVER procedure also promotes the Patch Edition to be the new Run Edition, but when called from the Run Edition that action is skipped. The table is now ready for use.

The CLEANUP procedure disables and removes the crossedition trigger.

3. Extract the updated table definition from your development database.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user> XYZ_USER_SERVICE
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password> XYZ_USER_SERVICE
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_USER_SERVICE
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_USER_SERVICE
   ```

   At last, you are ready to create the patch.

4. Create the patch.

   Patch Files:

   fnd/patch/115/xdf/xyz_user_service.xdf version 4

   fnd/patch/115/sql/XYZ_USER_SERVICE_X1.sql

   Manual apply phase actions for the file system:

   ```
   cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual apply actions for the database:

   For instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_service.xdf <apps_user>
   sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_USER_SERVICE_X1 APPLSYS
   sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_USER_SERVICE_F1
   ```

   For instances on R12.AD.C.Delta.7 and earlier:

   ```
   xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_user_service.xdf <apps_user>/<apps_password>
   sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_USER_SERVICE_X1 APPLSYS
   sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_USER_SERVICE_F1
   ```

5. Test the patch.

   When XDF applies the table update, the revised column and index is added, and the EV will be regenerated to use the new revised column. After the FCET is created and applied the revised column is populated with the new codes.

Multiple updates to the same table

Over time, you might need to make multiple changes to the structure or data of a particular table. Some of these change may require the use of additional revised columns and crossedition triggers. When the same table is patched with multiple crossedition triggers, the online patching tool requires that the order of trigger processing be explicitly defined. The previous FCET may have been applied in a previous patching session, but when multiple patches are merged into one rollup patch it will be the case that multiple crossedition for the same table are applied in the same patching cycle. In order to guarantee correct and predictable ordering of FCET processing, the database supports a "FOLLOWS" keyword in the trigger definition:

```
create or replace trigger XYZ_USER_SERVICE_F2
                before insert or update on &1..xyz_user_service
                for each row  forward crossedition
                FOLLOWS XYZ_USER_SERVICE_F1
                disable
...
```

When multiple changes are made to the same table, each new FCET must be defined as following the previous FCET:

1. F1
2. F2 follows F1
3. F3 follows F2
4. ... and so on ...

To continue our example, lets imagine that we are adding a new service level to our app and existing GOLD customers will automatically be promoted to the new level.

```
  XYZ_USER_SERVICE
           Name                                      Null?    Type
 ----------------------------------------- -------- --------------
 USER_ID                                   NOT NULL NUMBER
   -- PK, FK to FND_USER.USER_ID
 SERVICE_TYPE                              NOT NULL VARCHAR2(8)
   -- 'BRONZE'   - cheap service
   -- 'SILVER'   - plus service
   -- 'GOLD'     - best service
==>-- 'PLATINUM' - VIP
 COMMENTS                                          VARCHAR2(1000)
 SERVICE_STATUS                            NOT NULL VARCHAR2(8)
   -- 'ENABLED'  - service is active
   -- 'DISABLED' - service is not active.
```

1. Create the new column in your development database.

   Again, we use a new revised column to hold the new service type values. Add a new revised column manually in your development database.

   ```
   alter table APPLSYS.XYZ_USER_SERVICE
                 add (SERVICE_TYPE#2 varchar2(8) default '*NULL*' not null)
   /

   exec ad_zd_table.patch('APPLSYS', 'XYZ_USER_SERVICE')
   ```

2. Create a Forward Crossedition Trigger to populate the new column.

   The FCET loads the new column with upgrade service type codes. The logic will promote existing GOLD customers to VIP level. Note that the logic in this FCET depends on XYZ_USER_SERVICE_F1 having run already. We indicate this dependency using the FOLLOWS clause.

   ```
   REM ---- Create FCET ----
   REM dbdrv: sql ~PROD ~PATH ~FILE \
   REM dbdrv:     none none none sqlplus &phase=ccet \
   REM dbdrv:     checkfile:~PROD:~PATH:~FILE &un_fnd
   REM ---- Apply FCET ----
   REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
   REM dbdrv:   none none none sqlplus &phase=acet \
   REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet XYZ_USER_SERVICE_F2

   REM Copyright (c) 2013 Oracle Corporation, All Rights Reserved
    REM $Header$
   REM XYZ_USER_SERVICE_X2.sql
   REM     Populate XYZ_USER_SERVICE.SERVICE_TYPE

   SET VERIFY OFF;
   WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
   WHENEVER OSERROR EXIT FAILURE ROLLBACK;

   create or replace trigger XYZ_USER_SERVICE_F2
           before insert or update on &1..XYZ_USER_SERVICE
           for each row  forward crossedition
           follows XYZ_USER_SERVICE_F1 /* notice! */
           disable
   begin
           if :new.service_type#1 = 'GOLD' then
                   :new.service_type#2 := 'VIP';
           else
                   :new.service_type#2 := :new.service_type#1;
           end if;
   end;
   /

   commit;
   exit;
   ```

   You can apply and test the FCET in the development database as follows:

   ```
   sqlplus <apps_user> @XYZ_USER_SERVICE_X2 APPLSYS
           # note: the trigger will create with compilation error, that is OK
   sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_USER_SERVICE_F2
   sqlplus <apps_user>
                   exec ad_zd.finalize
                   exec ad_zd.cutover
                   exec ad_zd.cleanup
           quit
   ```

   Note the following behavior: When the XYZ_USER_SERVICE_F2 trigger is initially created, it will not compile, because the trigger definition makes reference to XYZ_USER_SERVICE_F1 which no longer exists. This database behavior is corrected later by the APPLY procedure. When you run the AD_ZD_TABLE_APPLY script, the new "F2" trigger will be enabled and applied to all existing rows, but since the XYZ_USER_SERVICE_F2 is defined as following XYZ_USER_SERVICE_F1, the APPLY procedure will automatically create an empty XYZ_USER_SERVICE_F1 trigger to satisfy the reference and allow "F2" to compile.

3. Extract the updated table definition from your development database:

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user>  XYZ_USER_SERVICE
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>  XYZ_USER_SERVICE
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_USER_SERVICE
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_USER_SERVICE
   ```

4. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_user_service.xdf version 5
   - fnd/patch/115/sql/XYZ_USER_SERVICE_X2.sql

   Manual apply phase actions for the file system:

   ```
   cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual apply phase actions for the database:

   For instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_service.xdf <apps_user>
   sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_USER_SERVICE_X2 APPLSYS
   sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_USER_SERVICE_F2
   ```

   For instances on R12.AD.C.Delta.7 and earlier:

   ```
   xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_user_service.xdf <apps_user>/<apps_password>
   sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_USER_SERVICE_X2 APPLSYS
   sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_USER_SERVICE_F2
   ```

5. Test the patch.

   When XDF applies the table update, the revised column and index will be added, and the EV will be regenerated to use the new revised column. After the FCET is created and applied, the revised column will be populated with the new codes.

Migrate data to a new table

In some cases the database schema for a product is reorganized so that existing data rows must be moved to a newly delivered table. New or replacement tables might be required if you are making a change to data model cardinality (meaning a change to the number of rows used to store the same information). The editioning view and revised column technique allows the developer to change how entity attributes are stored in columns, but it does not allow for changing the number of rows in a existing table.

Replacement tables are delivered by XDF/ODF as usual, but these tables must be populated using forward crossedition triggers rather than traditional upgrade scripts. The following example shows how to deliver and populate a replacement table for an existing table.

1. Create the replacement table in your development database.

   This is done exactly as delivering any other new table. In our example, we will create a replacement table for the WF_ITEMS table called WF_ITEMS_NEW.

   ```
   create table APPLSYS.WF_ITEMS_NEW
                                       (
                                              NEW_ID         NUMBER(15)     not null,
                                              ITEM_TYPE      VARCHAR2(8)    not null,
                                              ITEM_KEY       VARCHAR2(240)  not null,
                                              USER_KEY       VARCHAR2(240)
                                       )
                                       tablespace APPS_TS_TX_DATA
   /

   -- new primary key
   create unique index APPLSYS.WF_ITEMS_NEW_U1
                   on APPLSYS.WF_ITEMS_NEW ( NEW_ID )
                   tablespace APPS_TS_TX_IDX
   /

   -- this index is used for data migration
   create unique index APPLSYS.WF_ITEMS_NEW_U2
   ```

```
                on APPLSYS.WF_ITEMS_NEW ( ITEM_TYPE, ITEM_KEY )
                tablespace APPS_TS_TX_IDX
/
```

Note that the replacement table must have indexing necessary for the data migration logic to efficiently map rows in source tables to the corresponding rows in the replacement table.

2. Create a LOAD_ROW procedure for the replacement table

   The insert and update statements that populate a replacement table during data migration must be run from a PL/SQL package, they CANNOT be directly run by the data migration trigger itself. The reason for this is complicated, just accept it for now.

   Create a PL/SQL package procedure that will run the insert or update statements on the replacement table. These are best combined into a single procedure called LOAD_ROW that will accept the required attributes and then insert or update the row in the replacement table as needed.

```
create or replace package body WF_ITEMS_NEW_PKG as
                procedure LOAD_ROW(
                        X_ITEM_TYPE  in varchar2,
                        X_ITEM_KEY   in varchar2,
                        X_USER_KEY   in varchar2
                ) is
                begin
                        insert into wf_items_new ( new_id, item_type, item_key, user_key )
                                values ( wf_items_new_s1.nextval, x_item_type, x_item_key, x_user_key );
                exception when dup_val_on_index then
                        update wf_items_new
                        set user_key = x_user_key
                        where item_type = x_item_type
                                and item_key  = x_item_key;
                end;
end;
```

3. Create the data migration trigger.

   The data migration trigger is a forward crossedition trigger installed on the source table. This trigger will be fired for every row in the source table, and will also fire for any changes to the source table made by the running application. The migration trigger will call the LOAD_ROW procedure for the replacement table.

   Once you migrate data from a source table into a replacement table, we expect that the source table is now obsolete and will no longer be patched. To indicate this, we use a standard name for data migration triggers which is

```
 <TABLE_NAME>_FX
```

   The 'X' indicates that this is the final crossedition trigger that will ever be created for that table.

   If the source table had any previous crossedition triggers defined for it, then remember to specify the "follows <previous_fcet>" clause when you define the data migration trigger.

   The "dbdrv" commands for a data migration trigger are just like an ordinary forward crossedition trigger with one exception. A forward crossedition trigger is normally applied in "&phase=acet". But a data migration trigger should be applied in "&phase=acet+1". By delaying the running of the data migration trigger we can ensure that any crossedition triggers on the destination table are already applied and therefore enabled.

```
REM ---- Create FCET ----
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=ccet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd
REM ---- Apply FCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet+1 \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet WF_ITEMS_FX

REM Copyright (c) 2016 Oracle Corporation, All Rights Reserved
REM $Header$
REM WF_ITEMS_FX.sql
REM Data Migration Example:
REM
REM Migrate rows in WF_ITEMS to replacement table WF_ITEMS_NEW
REM
REM Key mapping:
REM
REM     WF_ITEMS      (ITEM_TYPE,ITEM_KEY)
REM     WF_ITEMS_NEW  (ITEM_TYPE,ITEM_KEY)
REM
REM Data Mapping
REM                         -> WF_ITEM_NEW.NEW_ID
REM     WF_ITEMS.ITEM_TYPE  -> WF_ITEMS_NEW.ITEM_TYPE
REM     WF_ITEMS.ITEM_KEY   -> WF_ITEMS_NEW.ITEM_KEY
REM     WF_ITEMS.USER_KEY   -> WF_ITEMS_NEW.USER_KEY
REM
SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace trigger WF_ITEMS_FX
  before insert or update or delete on &1..WF_ITEMS
  for each row  forward crossedition
  /* follows <previous_fcet> */ disable
begin
  if inserting or updating then

    -- migrate data to new table
    wf_items_new_pkg.load_row(:new.item_type, :new.item_key, :new.user_key);

  elsif deleting then

    -- Replicate delete to new table
    delete from wf_items_new
    where item_type = :old.item_type
      and item_key  = :old.item_key;

  end if;
end;
/
commit;
exit;
```

   IMPORTANT NOTE: For a data migration trigger, the apply phase should be "&phase=acet+1" rather than "@phase=acet". This is to guarentee that any future crossedition triggers on the destination table are enabled already before the data migration trigger is applied.

Patching rules for the replacement table

Initially, your replacement table will be brand new and have no revised columns, but you may decide to patch the structure of the replacement table in the future. If this occurs, you must take care to ensure that the original data migration trigger and LOAD_ROW logic will continue to work with the new replacement table structure. In particular, if you revise a not null or unique indexed column in the replacement table, then you must implement a reverse crossedition trigger to populate the old column using the same technique described for Seed Data Tables in the section "Update a Not Null or Unique Indexed Column".

Section 1.4.3.2: Seed Data Tables

Seed Data Tables must implement a special feature to support Online Patching called Editioned Data Storage. This feature allows a single seed data table to hold multiple copies of the seed data: During online patching, the original seed data remains in use by the Run Edition, and a separate copy of the data can be created for the Patch Edition. Editioned Data Storage allows a loader to modify the Patch Edition copy of seed data without affecting the Run Edition. Seed data tables are created and patched like ordinary tables with a few extra rules that will be explained presently.

Is your table really a seed data table? The extra patching standards and runtime overhead associated with seed data tables are best avoided, so please make sure that you do not upgrade a table to be a seed data table unless it is truly necessary. A real seed data table has ALL of the following properties:

- The Table contains data that is "part of the application" and is delivered and maintained via application patching. More specifically:
  - You (the application developer) create and deliver data in the table.
  - You expect to maintain and patch the data along with the application code.
  - Typically, you have a seed data loader for the purpose of delivering data updates.
- The Table Data affects runtime application functionality or appearance
  - Meaning the data is used by the code, rather than simply handled by the code. The data controls how the application operates.
  - Business Reference data such as Parties or Products is not seed data.
- The Table Data is predominately application seed data.
  - There can be user-entered rows, but it is not appropriate to treat a high volume transaction table as a seed data table.
  - Sample transaction or setup data is not seed data.

Create a new Seed Data Table

In this example, we add a new seed data table to the application to hold "service type" information. The standard service types are created and maintained by application development, and so this table meets the definition of a seed data table. The logical table structure is as follows:

```
XYZ_SERVICE_TYPES
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SERVICE_TYPE                              NOT NULL VARCHAR2(8)
 SERVICE_PRIORITY                          NOT NULL NUMBER
 HOME_PAGE                                          VARCHAR2(30)
```

1. Create the initial table definition in your development database. This is done just like an ordinary table. Remember that seed data tables should be stored in the APPS_TS_SEED tablespace.

```
  create table APPLSYS.XYZ_SERVICE_TYPES
        (
```

```
      SERVICE_TYPE     VARCHAR2(8)    not null,
      SERVICE_PRIORITY NUMBER         not null,
      HOME_PAGE        VARCHAR2(30)
    )
    tablespace APPS_TS_SEED
  /

  create unique index APPLSYS.XYZ_SERVICE_TYPES_U1
    on APPLSYS.XYZ_SERVICE_TYPES ( SERVICE_TYPE )
    tablespace APPS_TS_SEED
  /

  exec ad_zd_table.upgrade('APPLSYS', 'XYZ_SERVICE_TYPES')
```

2. Upgrade table to support Editioned Data Storage.

   This is a new required step for seed data tables and is done by calling the AD_ZD_SEED.UPGRADE procedure.

```
 exec ad_zd_seed.upgrade('XYZ_SERVICE_TYPES')
```

   Now the table is officially a Seed Data Table. The AD_ZD_SEED.UPGRADE procedure added a new column to the table key that stripes the data by edition, along with various supporting objects.The loader for a seed data table must be coded to call the AD_ZD_SEED.PREPARE procedure before changing the content of the table in an Online Patch, but no special action is required to update the table from the Run Edition.For this example, we can just put in some sample data directly.

```
  insert into XYZ_SERVICE_TYPES (service_type, service_priority, home_page)
    values ('VIP',    0, 'VIP_HOME');
  insert into XYZ_SERVICE_TYPES (service_type, service_priority, home_page)
    values ('GOLD',   1, 'GOLD_HOME');
  insert into XYZ_SERVICE_TYPES (service_type, service_priority, home_page)
    values ('SILVER', 2, 'SILVER_HOME');
  insert into XYZ_SERVICE_TYPES (service_type, service_priority, home_page)
    values ('BRONZE', 3, 'BRONZE_HOME');
  commit;
```

3. Create a loader for the seed data table:

   The Seed Data Loader is typically implemented as a loader configuration file (LCT) for the FNDLOAD generic loader. This is created in the usual way, with one new addition, which is to define the tables that must be prepared when seed data is patched using the loader. Here is a simple LCT definition for the example table:

```
-

COMMENT = "dbdrv: exec fnd bin FNDLOAD bin &phase=daa checkfile:~PROD:~PATH:~FILE &ui_apps 0 Y UPLOAD @FND:patch/115/import/xyzst.lct @~PROD:~PATH/~FILE"

DEFINE XYZ_SERVICE_TYPE
  KEY  SERVICE_TYPE     VARCHAR2(8)
  BASE SERVICE_PRIORITY VARCHAR2(8)
  BASE HOME_PAGE        VARCHAR2(30)
END XYZ_SERVICE_TYPE

DOWNLOAD XYZ_SERVICE_TYPE
" select SERVICE_TYPE, to_char(SERVICE_PRIORITY), HOME_PAGE
  from XYZ_SERVICE_TYPES
  where (:SERVICE_TYPE is null or SERVICE_TYPE like :SERVICE_TYPE) "

UPLOAD XYZ_SERVICE_TYPE
" begin
    update XYZ_SERVICE_TYPES
      set SERVICE_PRIORITY = :SERVICE_PRIORITY,
          HOME_PAGE = :HOME_PAGE
      where SERVICE_TYPE = :SERVICE_TYPE;

    if SQL%NOTFOUND then
      insert into XYZ_SERVICE_TYPES
        ( SERVICE_TYPE, SERVICE_PRIORITY, HOME_PAGE )
        values (:SERVICE_TYPE, to_number(:SERVICE_PRIORITY), :HOME_PAGE);
    end if;
  end; "

PREPARE XYZ_SERVICE_TYPE
  TABLE XYZ_SERVICE_TYPES

-----
```

   Notice the PREPARE statement at the end of the file. This statement tells the loader to prepare the 'XYZ_SERVICE_TYPES' table before attempting to load data for the XYZ_SERVICE_TYPE entity.

4. Extract the table definition from your development database:

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
perl xdfgen.pl <apps_user>  XYZ_SERVICE_TYPES
```

   For all instances on R12.AD.C.Delta.7:

```
perl xdfgen.pl <apps_user>/<apps_password>  XYZ_SERVICE_TYPES
```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_SERVICE_TYPES
```

   For RAC instances on R12.AD.C.Delta.6 and earlier::

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_SERVICE_TYPES
```

5. Extract starting seed data from your development database:

```
 FNDLOAD <apps_user> 0 Y DOWNLOAD xyzst.lct xyzst_data.ldt XYZ_SERVICE_TYPE
```

   Enter the password when prompted.

6. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_service_types.xdf
   - fnd/patch/115/import/xyzst.lct
   - fnd/patch/115/import/US/xyzst_data.ldt

   Manual apply phase actions for the file system:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
cp fnd/patch/115/import/* $FND_TOP/patch/115/import
cp fnd/patch/115/import/US/* $FND_TOP/patch/115/import/US
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the passwords when prompted:

```
xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_service_types.xdf <apps_user>FNDLOAD <apps_user> 0 Y UPLOAD $FND_TOP/patch/115/import/xyzst.lct $FND_TOP/patch/115/import/US/xyzst_data.ldt
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.7 and earlier:

```
xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK
$FND_TOP/patch/115/xdf/xyz_service_types.xdf <apps_user>/<apps_password>
FNDLOAD <apps_user> 0 Y UPLOAD $FND_TOP/patch/115/import/xyzst.lct
$FND_TOP/patch/115/import/US/xyzst_data.ldt
```

7. Test the patch.

   When the patch is applied, XDF first creates the initial table, and then automatically calls AD_ZD_TABLE.UPGRADE. XDF also detects that the table definition is for a seed data table, and calls the AD_ZD_SEED.UPGRADE procedure to install supporting objects for Editioned Data Storage.

   Once the seed data table is in place, the FNDLOAD procedure will load data into the table. The loader will automatically call the AD_ZD_SEED.PREPARE procedure for table to be loaded (although in this case the procedure will not do anything as there is no run edition copy of seed data).

   When the patch is complete, verify that the seed data table definition and contents are as expected.

Update a NOT NULL or Unique Indexed Column

If you patch an existing column in a seed data table that either (a) has a NOT NULL constraint with no default value, or (b) is part of a unique index, there is a new requirement: In addition to writing the Forward Crossedition Trigger to populate your revised column: You also need to code a Reverse Crossedition Trigger to populate the original column when data is loaded in the Patch Edition. For example, suppose we need to increase the size of the SERVICE_TYPE column from 8 to 30 bytes:

```
     XYZ_SERVICE_TYPES
     Name                                      Null?    Type
     ----------------------------------------- -------- --------------
==>  SERVICE_TYPE                              NOT NULL VARCHAR2(30)
     SERVICE_PRIORITY                          NOT NULL NUMBER
     HOME_PAGE                                          VARCHAR2(30)
```

1. Create the revised column in your development database.

```
  alter table APPLSYS.XYZ_SERVICE_TYPES
        add (SERVICE_TYPE#1 varchar2(30) default '*NULL*' not null)
      /

  exec ad_zd_table.patch('APPLSYS', 'XYZ_SERVICE_TYPES')
```

2. Create a Forward Crossedition Trigger to populate the revised column.

This is identical to the technique used for ordinary tables. However, the SQL script that creates the FCET will also create and apply a Reverse Crossedition Trigger.

3. Create a Reverse Crossedition Trigger to populate the original column.

   The reverse crossedition trigger only fires when the table contents is changed from the patch edition (such as during seed data loading).The purpose of the of the Reverse Crossedition Trigger is to populate the old column in some way that satisfies the old NOT NULL or UNIQUE constraint. The following is an example of SQL script that combines creation of both the FCET and RCET for a revised unique indexed column.

```
REM ---- Create FCET+RCET ----
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=ccet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd
REM ---- Apply FCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet XYZ_SERVICE_TYPES_F1
REM ---- Apply RCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:rcet XYZ_SERVICE_TYPES_R1

REM Copyright (c) 2013 Oracle Corporation, All Rights Reserved
REM $Header$
REM XYZ_SERVICE_TYPES_X1.sql
REM   Update XYZ_SERVICE_TYPES.SERVICE_TYPE to varchar2(30)

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

-- FCET Definition
create or replace trigger XYZ_SERVICE_TYPES_F1
  before insert or update on &1..XYZ_SERVICE_TYPES
  for each row  forward crossedition
  disable
begin
  :new.service_type#1 := :new.service_type;
end;
/

-- RCET Definition
create or replace trigger XYZ_SERVICE_TYPES_R1
  before insert or update on &1..XYZ_SERVICE_TYPES
  for each row  reverse crossedition
  disable
begin
  :new.service_type := substrb(:new.service_type#1, 1, 8);
end;
/

commit;
exit;
```

   You might have noticed that the example reverse crossedition trigger logic may not satisfy the uniqueness constraint of the old column for new data that is longer than 8 bytes. In cases where this is a concern, you can populate the old column with values from a sequence number, converted to an 8-byte string. The reverse crossedition trigger does not actually need to populate meaningful data in the old columns, it only needs ensure that database constraints on the old column are satisfied when rows are loaded in the patch edition.

   You can apply and test the FCET/RCET triggers in a development database as follows:

```
    sqlplus <apps_user> @XYZ_SERVICE_TYPES_X1 APPLSYS
    sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_F1
    sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_R1

    sqlplus <apps_user>
      exec ad_zd.finalize

      -- test insert into table
      insert into XYZ_SERVICE_TYPES
        (service_type, service_priority, home_page)
        values ('TEST_TYPE', 0, 'TEST');
      select * from applsys.xyz_service_types
        where service_type#1 = 'TEST_TYPE';
      rollback;

      exec ad_zd.cutover
      exec ad_zd.cleanup
```

   Since the reverse crossedition trigger is not applied to existing rows of the table, It is recommended that you make a test insert into the table in order to verify the trigger logic.

4. Extract the table definition from your development database:

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
perl xdfgen.pl <apps_user>  XYZ_SERVICE_TYPES
```

   For all instances on R12.AD.C.Delta.7:

```
perl xdfgen.pl <apps_user>/<apps_password>  XYZ_SERVICE_TYPES
```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_SERVICE_TYPES
```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_SERVICE_TYPES
```

5. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_service_types.xdf
   - fnd/patch/115/sql/XYZ_SERVICE_TYPES_X1.sql
   - fnd/patch/115/import/xyzst.lct

   Manual apply phase actions for the file system:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
cp fnd/patch/115/import/* $FND_TOP/patch/115/import
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
    xdfcmp.pl applsys/apps $FND_TOP/patch/115/xdf/xyz_service_types.xdf <apps_user>
    sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_SERVICE_TYPES_X1 APPLSYS
    sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_F1
    sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_R1
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.7 and earlier. Enter the password when prompted:

```
xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_service_types.xdf <apps_user>/<apps_password>
sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_SERVICE_TYPES_X1 APPLSYS
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_F1
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_R1
```

6. Test the patch.

   When applied to a patch edition, XDF creates the new revised column and index. Then the crossedition triggers will be installed. The FCET will populate the new revised column. The RCET will populate the old column if any data is loaded into the table in the patch edition. After cutover, the old index and crossedition triggers will be removed.

Multiple updates to the same seed data table

As mentioned in the preceding section for Tables, when you make second and subsequent updates to the same table, each new Forward Crossedition Trigger must be created so that it FOLLOWS the previous FCET. This rule applies to seed data tables as well, but seed data tables have an additional requirement to specify ordering for the second and subsequent Reverse Crossedition Triggers. Each new RCET must be created so that it PRECEDES the previous RCET, if any. The PRECEDES syntax looks like this:

```
 create or replace trigger XYZ_SERVICE_TYPES_R2
               before insert or update on &1..xyz_user_service
               for each row  reverse crossedition
               PRECEDES XYZ_SERVICE_TYPES_R1
               disable
...
```

So each new FCET FOLLOWS the previous FCET.

1. F1
2. F2 follows F1 ...
3. F3 follows F2 ...
4. ... and so on ...

And each new RCET PRECEDES the previous RCET.

1. R1

2. R2 precedes R1 ...
3. R3 precedes R2 ...
4. ... and so on ...

For our coding example, let us simply make a null change to the primary key column to demonstrate the technique:

```
XYZ_SERVICE_TYPES
    Name                                      Null?    Type
    ----------------------------------------- -------- ----------------
    SERVICE_TYPE                              NOT NULL VARCHAR2(30)
    SERVICE_PRIORITY                          NOT NULL NUMBER
    HOME_PAGE                                          VARCHAR2(30)
```

1. Create the revised column in your development database.

```
    alter table APPLSYS.XYZ_SERVICE_TYPES
                add (SERVICE_TYPE#2 varchar2(30) default '*NULL*' not null)
    /

    exec ad_zd_table.patch('APPLSYS', 'XYZ_SERVICE_TYPES')
```

2. Create a Forward Crossedition Trigger to populate the revised column.

   The new FCET must be created so that it FOLLOWS the previous FCET.

3. Create a Reverse Crossedition Trigger to populate the original column.

   The new RCET must be created so that it PRECEDES the previous RCET.

```
REM ---- Create FCET+RCET ----
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=ccet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd
REM ---- Apply FCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet XYZ_SERVICE_TYPES_F2
REM ---- Apply RCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:rcet XYZ_SERVICE_TYPES_R2

REM Copyright (c) 2013 Oracle Corporation, All Rights Reserved
REM $Header$
REM XYZ_SERVICE_TYPES_X2.sql
REM   Pretend update to XYZ_SERVICE_TYPES.SERVICE_TYPE attribute

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

-- FCET Definition
create or replace trigger XYZ_SERVICE_TYPES_F2
  before insert or update on &1..xyz_service_types
  for each row  forward crossedition
  follows XYZ_SERVICE_TYPES_F1
  disable
begin
  :new.service_type#2 := :new.service_type#1;
end;
/

-- RCET Definition
create or replace trigger XYZ_SERVICE_TYPES_R2
  before insert or update on &1..xyz_service_types
  for each row  reverse crossedition
  precedes XYZ_SERVICE_TYPES_R1
  disable
begin
  :new.service_type#1 := :new.service_type#2;
end;
/

commit;
exit;
```

4. Apply and test the FCET/RCET triggers in a development database as follows:

```
sqlplus <apps_user> @XYZ_SERVICE_TYPES_X2 APPLSYS
      # both triggers create with compilation errors, no worries.
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_F2
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_R2

sqlplus <apps_user>
  exec ad_zd.finalize

  -- test insert into table
  insert into XYZ_SERVICE_TYPES
     (service_type, service_priority, home_page)
     values ('TEST_TYPE', 0, 'TEST');
  select * from applsys.xyz_service_types
     where service_type#1 = 'TEST_TYPE';
  rollback;

  exec ad_zd.cutover
  exec ad_zd.cleanup
```

5. Extract the table definition from your development database:

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure::

```
perl xdfgen.pl <apps_user> XYZ_SERVICE_TYPES
```

   For all instances on R12.AD.C.Delta.7:

```
perl xdfgen.pl <apps_user>/<apps_password> XYZ_SERVICE_TYPES
```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_SERVICE_TYPES
```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_SERVICE_TYPES
```

6. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_service_types.xdf
   - fnd/patch/115/sql/XYZ_SERVICE_TYPES_X2.sql

   Manual apply phase actions for the file system:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the passwords when prompted:

```
xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_service_types.xdf <apps_user>

sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_SERVICE_TYPES_X2 APPLSYS
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_F2
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_R2
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.7 and earlier:

```
xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_service_types.xdf <apps_user>/<apps_password>

sqlplus <apps_user> @$FND_TOP/patch/115/sql/XYZ_SERVICE_TYPES_X2 APPLSYS
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_F2
sqlplus <apps_user> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_SERVICE_TYPES_R2
```

Section 1.4.3.3: Temporary Tables

A global temporary table is a table that does not have permanent storage. Rows in a temporary table are held in memory either for a single transaction or for a single session, and are not accessible outside of the current session. Temporary tables are normally used to hold interim or summarized results in order to improve the performance of some other processing.

A global temporary table is a non-editioned object with a special restriction compared to ordinary tables: A temporary table cannot be modified in any way during online patching. This is a database restriction: attempting to modify a temporary table while it is in use by any other session will result in an oracle error such as "ORA-14450: attempt to access a transactional temp table already in use". Therefore, patching a temporary table definition requires a special procedure.

Section 1.4.3.3.1 Create a Temporary Table

To create and deliver the first version of a temporary table, you create the table in your development environment, extract it using XDF, and include the XDF in your patch. This process should be the same as it was in classic downtime patching.

1. Create the temporary table in a development database. This is done using standard SQL*Plus. For example:

```
create global temporary table APPLSYS.XYZ_USER_GT
      (
        USER_ID         NUMBER(15)     not null,
        USER_DATA       VARCHAR2(10)
      )
    /

    create index APPLSYS.XYZ_USER_GT_N1 on APPLSYS.XYZ_USER_GT (USER_ID)
    /

    create or replace synonym XYZ_USER_GT for APPLSYS.XYZ_USER_GT2;
```

   Note that the temporary table must be created in a product schema, not directly in the APPS schema. The APPS schema must contain a synonym that points to the temporary table, which will serve as the permanent logical name of the temporary table. When the temporary table is be patched in the future, you will create a new temporary table with a different name but the logical name (APPS synonym) will stay the same.

   It is possible to create indexes on a temporary table. The index information will be included when you extract the definition with XDF.

   After manual creation, you should validate that the temporary table definition is correct and works as expected.

2. Extract the temporary table definition to an XDF file. This is done as follows:

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
perl xdfgen.pl  <apps_user> XYZ_USER_GT
```

   For all instances on R12.AD.C.Delta.7:

```
perl xdfgen.pl  <apps_user>/<apps_password> XYZ_USER_GT
```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl  <apps_user>/<apps_password>@<DB_SID> XYZ_USER_GT
```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_USER_GT
```

3. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_user_gt.xdf

   Manual apply phase actions for the file system:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
 xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_gt.xdf <apps_user>
```

   Manual apply phase actions for the database: for instances on R12.AD.C.Delta.7 and earlier:

```
 xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK
$FND_TOP/patch/115/xdf/xyz_user_gt.xdf <apps_user>/<apps_password>
```

Section 1.4.3.3.2 Revise an Existing Temporary Table

When making any structural change to a temporary table or its indexes, it is not possible to alter the existing temporary table in place. Instead, you must create a new temporary table with a different name, and change the existing APPS synonym to point at the new table.

1. Create the revised temporary table in a development database. This is done using standard SQL*Plus. For example

```
create global temporary table APPLSYS.XYZ_USER_GT2
      (
        USER_ID         NUMBER(15)     not null,
        JOB_TYPE        VARCHAR2(8)    not null,
        USER_DATA       VARCHAR2(10)
      )
    /

    drop index APPLSYS.XYZ_USER_GT_N1;
    create index APPLSYS.XYZ_USER_GT_N1
      on APPLSYS.XYZ_USER_GT2 (USER_ID, JOB_TYPE);

    create or replace synonym XYZ_USER_GT for APPLSYS.XYZ_USER_GT2;
```

   You can keep indexes with their original names, even though the name of the underling temporary table has changed.

   After this manual creation, you should validate that the revised temporary table definition is correct and works as expected.

2. Extract the temporary table definition to an XDF file using the original file name.

   Run xdfgen.pl on the APPS synonym for the temporary table. Do not use the actual temporary table name. By using the APPS synonym, you allow XDF to record information about the link between the synonym and the new temporary table.

   Warning: If your instance is on R12.AD.C.Delta.6 or earlier, you will need to ensure that a temporary table with a name matching the APPS synonym exists in addition to the revised temporary table that you are trying to extract. You do not need to do this with R12.AD.C.Delta.7 or later.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
perl xdfgen.pl <apps_user> XYZ_USER_GT
```

   For all instances on R12.AD.C.Delta.7:

```
perl xdfgen.pl <apps_user>/<apps_password> XYZ_USER_GT
```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID> XYZ_USER_GT
```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

```
perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name>  XYZ_USER_GT
```

   When the extracted XDF is applied to a target system, XDF will automatically do the following:

   - Create the revised temporary table
   - Create the revised indexes using an alternate name
   - Change the original APPS table synonym to point to the revised temporary table in the patch edition.
   - Submit a deferred CLEANUP DDL to drop the old temporary table during the cleanup phase.

3. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_user_gt.xdf

   Manual apply phase actions for the file system:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

```
 xdfcmp.pl <applsys_user> $FND_TOP/patch/115/xdf/xyz_user_gt.xdf <apps_user>
```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.7 and earlier:

```
 xdfcmp.pl <applsys_user>/<applsys_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_user_gt.xdf <apps_user>/<apps_password>
```

4. Test the patch.

   When the XDF is applied to the patch edition, XDF will create the revised temporary table and index (using an alternate name for the indexes). XDF will then change the Patch Edition APPS table synonym to point to the revised temporary table. In the Run Edition, the APPS table synonym continues to point at the original temporary table, which remains undisturbed. XDF also submits a deferred CLEANUP DDL to drop the old temporary table during the cleanup phase. During cutover, the old index is dropped and the new index is renamed to the specified name. During CLEANUP the deferred DDL is run to drop the old temporary table.

Section 1.4.3.4: Materialized Views

In Oracle Database Release 11g Release 2 (11.2), a materialized view definition may not reference editioned objects. However, application developers must define the materialized view query in terms of APPS table synonyms and views which are editioned objects. To work around the restriction on materialized views, the Oracle E-Business Suite Online Patching solution supports a new Effectively-Editioned Materialized View compound object. The developer-specified query is stored in an ordinary view, called the Logical View. The materialized view is then generated from the logical view, using a new database feature that translates the logical query into an equivalent "implementation query" which depends only on non-editioned objects.

In Oracle Database 12c Release 1 (Release 12.1) and later it is possible to create a native materialized view definition that references editioned objects by using the "EVALUATE USING CURRENT EDITION" clause in the create statement. Customers on Oracle Database 12.1 or later can use this approach to create custom materialized views, and in doing so, can avoid the special development procedures described immediately below. Instead, they can review the subsequent section Creating and Changing Custom Materialized Views on Oracle Database 12c. Consult the *Oracle Database SQL Language Reference* for information on this feature. The Oracle E-Business Suite Online Patching tool will automatically alter the evaluation edition of materialized views to the run edition during online patching cutover. Note: Oracle E-Business Suite shipped products support operation on Oracle Database Release 11.2 and thus do not use the evaluation edition feature.

This section describes how to work with the effectively-editioned materialized view compound object.

1.4.3.4.1 Creating a New Materialized View

On an editioned database, you can no longer create a materialized view directly.

1. Create the Logical View in your development database.

   The Logical View is an ordinary database view that implements the desired query. The Logical View name must be the desired materialized view name with a '#' character appended to it.In this example, we intend to create a materialized view called XYZ_SCHEMAS_MV that presents some information about the database schemas associated with Oracle E-Business Suite.We start by creating the logical view XYZ_SCHEMAS_MV#:

   ```
   create or replace view XYZ_SCHEMAS_MV# as
         select   upper(oracle_username) USERNAME,
                  decode(read_only_flag,
                    'C', 'pub', 'E', 'applsys', 'U', 'apps') USERTYPE
         from     fnd_oracle_userid
         where    read_only_flag in ('C', 'E', 'U');
   ```

   While it is acceptable for the logical view to depend on editioned synonyms and views, it must not depend on editioned PL/SQL functions, such as those in the Oracle E-Business Suite APPS schema (built-in PL/SQL functions such as "upper" are acceptable). Test the Logical View to ensure that its shape and results are correct.

   ```
   select * from XYZ_SCHEMAS_MV#;

   USERNAME                       USERTYP
   ------------------------------ -------
   APPLSYS                        applsys
   APPS                           apps
   APPLSYSPUB                     pub
   ```

2. Generate the Materialized View.

   On an editioned database, materialized views are generated from their corresponding logical views using the AD_ZD_MVIEW.UPGRADE procedure.

   ```
   exec ad_zd_mview.upgrade('APPS', 'XYZ_SCHEMAS_MV')
   ```

   In this example, the UPGRADE procedure detects that materialized view is missing and generates it from the Logical View. The Materialized View definition is generated by transforming the Logical View query into an equivalent implementation query that directly references the underlying tables and columns. You can see the resulting MV implementation objects using the "ADZDSHOWMV" utility script:

   ```
   sqlplus <apps_user> @ADZDSHOWMV XYZ_SCHEMAS_MV
   -- MV Objects
   OBJECT_NAME                    OBJECT_TYPE         STATUS    DESCRIPTION
   ------------------------------ ------------------  --------- --------------------
   XYZ_SCHEMAS_MV                 MATERIALIZED VIEW   VALID      Materialized View
   XYZ_SCHEMAS_MV                 TABLE               VALID      Container Table
   XYZ_SCHEMAS_MV#                VIEW                VALID      Logical View

   -- MV Properties
   MV_NAME                        REFERS REFRESH_ STALENESS
   ------------------------------ ------ -------- --------------
   XYZ_SCHEMAS_MV                 DEMAND FORCE    FRESH
   ```

   The MV implementation query should never be changed directly. It must always be generated from the logical view using the UPGRADE procedure. The MV implementation query can be difficult to read and normally the developer will not need to look at it. But it is worth examining the implementation query of our example to understand what the transformation is doing. The formatted MV implementation query for our example logical view is as follows:

   ```
   CREATE MATERIALIZED VIEW "APPS"."XYZ_SCHEMAS_MV"
     ("USERNAME", "USERTYPE") AS
   SELECT
     UPPER("A1"."ORACLE_USERNAME") "USERNAME",
     DECODE("A1"."READ_ONLY_FLAG",
            'C','pub','E','applsys','U','apps') "USERTYPE"
   FROM "APPLSYS"."FND_ORACLE_USERID" "A1"
   WHERE "A1"."READ_ONLY_FLAG"='C'
      OR "A1"."READ_ONLY_FLAG"='E'
      OR "A1"."READ_ONLY_FLAG"='U'
   ```

   Notice that while the logical view references the APPS FND_ORACLE_USERID table synonym, the materialized view references the base table directly. The generated MV is automatically maintained by online patching whenever the logical view or anything it depends on is changed in a patch. Once generated, you can query or refresh the MV as usual.

   ```
   select * from XYZ_SCHEMAS_MV;

   USERNAME                       USERTYP
   ------------------------------ -------
   APPLSYS                        applsys
   APPS                           apps
   APPLSYSPUB                     pub
   ```

3. Extract the MV definition using XDF.

   Once the generated MV has been tested, you can extract the definition using XDF. XDF has been extended to automatically substitute the MV Logical View query for the implementation query in extracting an MV definition.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user>  XYZ_SCHEMAS_MV
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>  XYZ_SCHEMAS_MV
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_SCHEMAS_MV
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_SCHEMAS_MV
   ```

4. Create the patch.

   Patch Files:

   - fnd/patch/115/xdf/xyz_schemas_mv.xdf

   Manual apply phase actions for the file system:

   ```
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
    xdfcmp.pl <apps_user> $FND_TOP/patch/115/xdf/xyz_schemas_mv.xdf
   ```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.7 and earlier:

   ```
   xdfcmp.pl <apps_user>/<apps_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_schemas_mv.xdf
   ```

   Note that unlike tables, the MV definition is applied in the APPS schema, so the xdfcmp.pl syntax is slightly different.

5. Test the patch.

   When the XDF file is applied on a target database, XDF will automatically create the Logical View and then use the UPGRADE procedure to generated the Materialized View on the target database. The generated materialized view query will vary depending on the definitions of the objects that the Logical View depends on in the specific target database. In this case, the created MV is new (does not already exist in the target database) so XDF creates the MV immediately. The case of changing an existing MV is covered in the next section.

1.4.3.4.2 Changing a Materialized View

To change an existing materialized view in a development database, the developer will replace the Logical View with an updated definition and then regenerate the Materialized View implementation. The patching procedure is the same.

1. Replace the Logical View in your development database.

   To change the definition of a materialized view, you first replace the definition of the corresponding logical view in your development database. This is done using SQL DDL as usual. In this example, we create a new view that our MV definition will reference, and then update the XYZ_SCHEMAS_MV# Logical View to use the new view. When you are satisfied with the results of the updated Logical View, remember to call the AD_ZD_MVIEW.UPGRADE procedure to regenerate materialized view implementation.

   ```
   /* new view for schema type information - xyz_schema_types.sql */
       create or replace view XYZ_SCHEMA_TYPES as
         select lv.lookup_code  CODE
           ,    lv.meaning      MEANING
         from   fnd_lookup_values lv
   ```

```
  where  lv.lookup_type = 'ORACLEID_PRIVILEGE_INVIS'
    and  lv.language = 'US';

/* change XYZ_SCHEMAS_MV logical view to use XYZ_SCHEMA_TYPES view */
create or replace view XYZ_SCHEMAS_MV# as
  select fou.oracle_username USERNAME
     ,   st.meaning USERTYPE
  from   fnd_oracle_userid fou, xyz_schema_types st
  where  fou.read_only_flag in ('C', 'E', 'U')
    and  st.code = fou.read_only_flag;

/* test logical view as needed */
select * from XYZ_SCHEMAS_MV#;

/* regenerate materialized view implementation */
exec ad_zd_mview.upgrade('APPS', 'XYZ_SCHEMAS_MV')

/* test materialized view as needed */
select * from XYZ_SCHEMAS_MV;
```

During an online patching cycle, out-of-date MV regeneration happens automatically during the cutover phase. But when working in the run edition of a development database, you will need to perform MV regeneration yourself when you are ready using the AD_ZD_MVIEW.UPGRADE procedure.

2. Extract the MV definition.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user>  XYZ_SCHEMAS_MV
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>  XYZ_SCHEMAS_MV
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_SCHEMAS_MV
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_SCHEMAS_MV
   ```

3. Create the patch.

   Patch files:

   - fnd/patch/115/sql/xyz_schema_types.sql
   - fnd/patch/115/xdf/xyz_schemas_mv.xdf

   Manual apply phase actions for the file system:

   ```
   cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.8 and later. Enter the password when prompted:

   ```
   sqlplus <apps_user> @$FND_TOP/patch/115/sql/xyz_schema_types
   xdfcmp.pl <apps_user> $FND_TOP/patch/115/xdf/xyz_schemas_mv.xdf
   ```

   Manual apply phase actions for the database for instances on R12.AD.C.Delta.7 and earlier:

   ```
   sqlplus <apps_user> @$FND_TOP/patch/115/sql/xyz_schema_types
   xdfcmp.pl <apps_user>/<apps_password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_schemas_mv.xdf
   ```

1.4.3.4.3 Creating and Changing Custom Materialized Views on Oracle Database 12c

The Oracle Database 12c Release 1 (12.1) and later removes the restriction that materialized views cannot reference editioned objects. Customers can use the "Evaluation Edition" feature to develop custom materialize views using native Oracle DDL.

Note: Oracle E-Business Suite products support operations on Oracle Database Release 11.2 and therefore do not use the evaluation edition feature.

It is important to note the following when creating custom materialized views on Oracle Database 12c:

- Materialized views are still non-editioned objects in Oracle Database 12c and cannot depend on editioned objects unless they mention an Evaluation Edition in which names of editioned objects are resolved. This feature allows materialized views to reference editioned objects, but it **does not** make the materialized views editioned objects themselves. Dropping or invalidating a materialized view in the Patch Edition will still invalidate any object that depends on the materialized view in the Run Edition. Therefore maintenance of custom materialize views must be done either **during or after** the cutover phase.
- The USING TRUSTED CONSTRAINTS clause enables you to create a materialized view on top of a table that has a non-NULL Virtual Private Database (VPD) policy on it. Materialized view results are computed based on the rows and columns filtered by VPD policy. All Oracle E-Business Suite seed data tables have a VPD defined on the associated editioning view which means that all custom materialized views that join to Oracle E-Business Suite seed data tables must use the USING TRUSTED CONSTRAINTS clause.
- You will need to use a custom script to create the materialized view in the production instance because xdfgen does not currently support materialized views that use the EVALUATION EDITION clause.

This section describes how to work with native Oracle DDL to created custom materialized views.

Create a Custom Materialized View on Oracle Database 12c

The following example illustrates how to create an equivalent materialized view XYZ_SCHEMAS using native DLL:

```
create materialized view XYZ_SCHEMAS
build immediate
refresh on demand
evaluate using CURRENT EDITION as
  select  upper(oracle_username) USERNAME,
          decode(read_only_flag,
            'C', 'pub', 'E', 'applsys', 'U', 'apps') USERTYPE
  from   fnd_oracle_userid
  where  read_only_flag in ('C', 'E', 'U');
```

To create a materialized view that joins to an Oracle E-Business Suite seed data table or any table/editioning view with a VPD policy, the USING TRUSTED CONSTRAINTS clause must be defined:

```
create materialized view XYZ_SCHEMAS
build immediate
refresh on demand using trusted constraints
evaluate using CURRENT EDITION as
 select   upper(fou.oracle_username) USERNAME,
            decode(fou.read_only_flag,
              'C', 'pub', 'E', 'applsys', 'U', 'apps') USERTYPE,
            flv.meaning ENABLED
 from    fnd_oracle_userid fou,
           fnd_lookup_values flv
 where fou.read_only_flag in ('C', 'E', 'U')
 and    fou.enabled_flag = flv.lookup_code
 and    flv.view_APPLICATION_ID = 0
 and    flv.language = userenv('LANG')
 and    flv.lookup_type = 'YES/NO';
```

Note: If the USING TRUSTED CONSTRAINTS clause is omitted from the materialized view definition the following error will be raised when attempting to create the materialized view: "ORA-30372: fine grain access policy conflicts with materialized view".

Change a Custom Materialized View on Oracle Database 12c

This section applies to instances on Oracle Database 12c Release 1 (12.1) and later. Changes to existing materialized views MUST be done either during or after the cutover phase. Changing a materialized view while the Oracle E-Business Suite is online may result in errors in the running application.

- During cutover: Submit a deferred CUTOVER DDL to drop and recreate the Materialized view during the cutover phase.

  For example:

  ```
  exec ad_zd.load_ddl('CUTOVER','drop materialized view XYZ_SCHEMAS')
  exec ad_zd_load_ddl('CUTOVER', 'create materialized view XYZ_SCHEMAS build deferred refresh on demand using trusted constraints evaluate using current edition as select ~* from fnd_oracle_user_id
  ```

  Note: It is good practice to specify "build deferred" to ensure cutover processing is not delayed by building the materialized view.

- After cutover:

  Manually drop and recreate the custom materialized using custom scripts.

1.4.3.4.4 Materialized View Online Patching - how it works

An effectively-editioned Materialized View includes both a Logical View (managed by the developer) and a Materialized View (generated by Online Patching). The Logical View is an ordinary database view, and is therefore an editioned object that can be changed in the Patch Edition without affecting the Run Edition. But the generated Materialized View is a non-editioned object, meaning the definition and content of the materialized view is shared across all editions. In order to avoid breaking the running application during an online patch, the system must defer materialized view regeneration until the cutover phase, when the application is down.

During online patching, materialized view regeneration happens automatically during the cutover phase whenever the materialized view implementation is out-of-date with respect to the Logical View. A materialized view implementation becomes out-of-date if

- the Logical View is patched
- anything which the Logical View depends on is patched or recompiled
- any seed data table the materialized view depends on is prepared for loading new content in the patch edition

Materialized view cutover processing will take care to avoid regenerating materialized views unless there is an actual change required in the MV implementation. When regenerating the MV implementation, the system will try to preserve the existing MV container table (with its data), but if the shape of the container table must change, then the container table is dropped and recreated automatically.

Section 1.4.4: Advanced Topics

Section 1.4.4.1 Data Fixer Patch

In some cases, the developer must produce a patch for the purpose of fixing problems in application data while leaving the application definition itself unchanged. Examples of data problems include:

- Referential integrity problems (orphan detail records, broken FK references)
- Improper data values (for example, due to incomplete validation logic)
- Junk records, duplicate records, unnecessary backup records
- Leftover temporary records or generated temporary objects
- Problems introduced through incorrect migration logic or configuration
- Damage from incorrectly coded customizations or other user error

The common theme is that the data fixer patch:

- Fixes application data to be valid for the existing application definition
- Does not change the application definition

Since the data fixer logic operates on existing application data, it must run in the Run Edition of the system rather than the Patch Edition (this is because DML operations on existing data are not allowed in the patch edition except on prepared seed data tables). To be run safely in the Run Edition the data fixer logic must meet the following requirements:

- The processing must not make any schema changes to existing tables that are in use by the running application.

  It is acceptable to drop tables and other objects that are not used by the application, such as backup and temporary objects.
- The processing must not change the existing application definition (and therefore should not invalidate views, packages, or other code that is in use by the running application).
- The processing must not make incompatible changes to application data.
  - It is acceptable to delete invalid or unused data that is not referenced by the running application.
  - It is acceptable to update existing data if the update is compatible with existing application logic. Be careful to only update those rows that actually need changes, to minimize locking conflicts with the running application.
  - It is acceptable to insert missing data.
- The processing must operate correctly against the effective 12.2 baseline which is 12.2.3.

1. 1. Create Data Fixer logic in your development database.

   Data fixer logic usually takes the form of SQL scripts that include DML or anonymous PL/SQL blocks which scan existing data for problems and insert/update/delete as needed to fix those problems. This logic can be developed and tested in the run edition of your development database. When the data fixer logic is delivered via a patch, you have two choices for how that logic will be carried out:

   - If the logic requires human input in order to run, then define the scripts with "dbdrv: none". This tells adpatch not to run the scripts, but simply to copy them to the target system. In this case, your patch readme file should document the manual steps for how a user should then run the scripts to fix their data.
   - If the logic can be run automatically, then define the script with appropriate "dbdrv: sql ..." command and include the "exec ad_zd.set_edition('RUN') command at the beginning of the script. The dbdrv command causes adpatch to run the script during the patch apply. The ad_zd.set_edition command causes the script to switch to the Run Edition before running the fixer logic.

   To make the data fixer logic run automatically in the Run Edition even when it is invoked during Online Patching we must include a command at the top of the script that switches the session context to the Run Edition. This command is as follows:

   ```
   exec ad_zd.set_edition('RUN')
   ```

   This command allows the script to be included in an online patch. The script will initially be started in the Patch Edition (like all Online Patching actions), but it will then immediately switch to the Run Edition before running the data fixer logic.

   Since a Data Fixer Patch does not change the existing application definition, it can be safely applied in any patching mode (online, downtime, or hotpatch). This is one of the few types of patches that can be applied in hotpatch mode, so make sure to documented in the patch readme that the patch can be applied in hotpatch mode if desired. It is not necessary to use hotpatch mode, of course, any patch that is safe for hotpatch can also be applied in an ordinary online patching cycle.

   As an example, we will create a data fixer patch that deletes profile option values which do not match an existing profile option definition (an example of the orphaned details rows problem). Since no user input is required, we will code the script to run automatically during patch apply.

   ```
   REM dbdrv: sql ~PROD ~PATH ~FILE none none none sqlplus &phase=upg \
   REM dbdrv:   checkfile:~PROD:~PATH:~FILE
   REM Copyright (c) 2014 Oracle Corporation, All Rights Reserved
   REM $Header$
   REM @xyz_povfix.sql
   REM Deletes orphaned rows from FND_PROFILE_OPTION_VALUES.

   SET VERIFY OFF;
   WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
   WHENEVER OSERROR EXIT FAILURE ROLLBACK;

   REM Switch to Run Edition
   exec ad_zd.set_edition('RUN')

   REM Delete orphan profile option values
   delete from fnd_profile_option_values pov
   where not exists
   ( select APPLICATION_ID, PROFILE_OPTION_ID
   from fnd_profile_options po
   where po.APPLICATION_ID = pov.APPLICATION_ID
   and po.PROFILE_OPTION_ID = pov.PROFILE_OPTION_ID )
   /

   commit;
   exit;
   ```

2. Create the patch.

   Patch files:

   - fnd/patch/115/sql/xyz_povfix.sql

   Manual Apply actions for File System:

   ```
   cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
   ```

   Manual Apply actions for Database:

   ```
   sqlplus <APPS username> @$FND_TOP/patch/115/sql/xyz_povfix
   ```

3. Test the patch.

   To apply a data fixer patch as a manual patch, simply run the file system apply and database apply commands against the target database.

   ```
   apply_fs.sh
   apply_db.sh
   ```

Section 1.4.4.2 Partition Exchange

The Oracle Database supports a feature called "Partition Exchange", which allows data partitions to be swapped between two tables with identical structures. This feature is sometimes used by application products to allow the background creation of a large data set that can then be quickly "swapped in" to the main runtime table that is used by the application.

If your application does not use partition exchange, you can skip this section. Before reading further about the implications of partition exchange under Online Patching, please ensure that you are familiar with partitioning concepts and technology. Refer to the Oracle Database documentation for more information on partitioning.

When two tables have identical structure, it is possible to exchange a data partition in one table with that of another. This partition exchange is a quick data dictionary operation, and does not actually copy rows around, so it is very fast. For this reason, partition exchange can be used to implement a staging table architecture where a lengthy loading or generation process puts data into a staging table that is not visible to the normal application runtime, and then when loading is complete the staged data can be instantly swapped into the runtime table. The partition exchange feature works under online patching, but there are some additional rules that must be followed for the feature to work safely.

Create a Main and Stage table pair

To implement a solution based on partition exchange you will have one main table and one or more stage tables. The main table is used by application runtime and may have any number of partitions. The stage table must have exactly the same column structure as the main table.

1. Create the Main and Stage table pair in your development database

   In this example, we have a main table with several partitions, only one of which will actually be swapped with a staging table. As usual, we create the initial tables in the run edition of our development database using SQL*Plus. First we create the main table that is used by the application runtime:

   ```
   -- Main Table
   create table APPLSYS.XYZ_TASKS
     (
       USER_ID     number(15),
   ```

```
    TASK_TYPE   varchar2(8),    /* OPEN, CLOSED */
    TASK_INFO   varchar2(30),
    SOURCE      varchar2(8)     /* LOCAL, SYNCED */
  )
  tablespace APPS_TS_TX_DATA
  partition by list(source)
  (
    partition LOCAL   values('LOCAL'),
    partition SYNCED  values('SYNCED'),
    partition OTHER   values(DEFAULT)
  )
/

create index APPLSYS.XYZ_TASKS_N1
  on applsys.xyz_tasks (user_id) local
  tablespace APPS_TS_TX_IDX
/

exec ad_zd_table.upgrade('APPLSYS', 'XYZ_TASKS')

insert into xyz_tasks (user_id, task_type, task_info, source)
  values (0, 'OPEN', 'take out trash', 'LOCAL');
insert into xyz_tasks (user_id, task_type, task_info, source)
  values (0, 'OPEN', 'vacuum the house', 'SYNCED');
commit;
```

Note that the main table has three partitions based on the SOURCE column. In our example, data in the 'SYNCED' partition will be swapped in from a staging table that is loaded by some external process. Also notice that the index on the table is "local" meaning that the index inherits the same partitioning scheme as the main table. This allows the index information to be swapped along with the table data when the exchange is run.

We then proceed to create the staging table in the development database:

```
-- Staging Table (must have exactly same structure as Main Table)
create table APPLSYS.XYZ_TASKS_SYNC
  (
    USER_ID     number(15),
    TASK_TYPE   varchar2(8),    /* OPEN, CLOSED */
    TASK_INFO   varchar2(30),
    SOURCE      varchar2(8)     /* LOCAL, SYNCED */
  )
  tablespace APPS_TS_TX_DATA
/

create index APPLSYS.XYZ_TASKS_SYNC_N1
  on applsys.xyz_tasks_sync (user_id)
  tablespace APPS_TS_TX_IDX
/

exec ad_zd_table.upgrade('APPLSYS', 'XYZ_TASKS_SYNC')
```

The staging table must have identical physical structure to the main table. Remember that in an editioned database, tables have both a physical structure (the actual columns in the table) and a logical structure (the subset of of physical columns exposed through the editioning view). It is the physical table structure that must match exactly, the logical structure is not relevant to partition exchange.

With both tables in place, we can now verify simple partition exchange in the development environment.

```
delete from xyz_tasks_sync;
insert into xyz_tasks_sync (user_id, task_type, task_info, source)
  values (0, 'OPEN', 'pull weeds', 'SYNCED');
commit;

select * from xyz_tasks;

   USER_ID TASK_TYP TASK_INFO                      SOURCE
---------- -------- ------------------------------ --------
         0 OPEN     take out trash                 LOCAL
         0 OPEN     vacuum the house               SYNCED

alter table applsys.xyz_tasks
  exchange partition synced with table applsys.xyz_tasks_sync
  including indexes without validation;

select * from xyz_tasks;

   USER_ID TASK_TYP TASK_INFO                      SOURCE
---------- -------- ------------------------------ --------
         0 OPEN     take out trash                 LOCAL
         0 OPEN     pull weeds                     SYNCED
```

2. Extract the Main and Stage table definition from your development database:

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <apps_user> XYZ_TASKS
   perl xdfgen.pl <apps_user> XYZ_TASKS_SYNC
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <apps_user>/<apps_password> XYZ_TASKS
   perl xdfgen.pl <apps_user>/<apps_password> XYZ_TASKS_SYNC
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_TASKS
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID>  XYZ_TASKS_SYNC
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_TASKS
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_Instance_Name> XYZ_TASKS_SYNC
   ```

   Forever more, the main and stage table must be delivered and patched as a pair, so that their physical column structure remains identical.

3. Create the patch.

   Patch files:

   - fnd/patch/115/xdf/xyz_tasks.xdf
   - fnd/patch/115/xdf/xyz_tasks_sync.xdf

   Manual Apply actions for the File System:

   ```
   cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
   ```

   Manual Apply actions for the Database for instances on R12.AD.C.Delta.8 and later:

   ```
   xdfcmp.pl <APPLSYS username> $FND_TOP/patch/115/xdf/xyz_tasks.xdf <APPS username>
   xdfcmp.pl <APPLSYS username> $FND_TOP/patch/115/xdf/xyz_tasks_sync.xdf <APPS username>
   ```

   Manual Apply actions for the Database for instances on R12.AD.C.Delta.7 and earlier:

   ```
   xdfcmp.pl <APPLSYS username>/<APPLSYS password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_tasks.xdf <APPS username>/<APPS password>
   xdfcmp.pl <APPLSYS username>/<APPLSYS password>@$TWO_TASK $FND_TOP/patch/115/xdf/xyz_tasks_sync.xdf <APPS username>/<APPS password>
   ```

Patch a Main and Stage table pair

Whenever a partition exchange main table is patched, the related staging table must also be patched in exactly the same way. This includes any new or revised columns, indexes, and forward crossedition trigger logic. Advanced warning: even when the main and stage table are patched together, there will be a short period of time when the online patch apply is in progress where the main and stage table structure are temporarily out of sync (because the patch apply actions have not yet completed). We will discuss how to deal with this issue later.

1. Alter the main and stage table definitions in your development database.

   To demonstrate patching the main and stage table together, we will simply extend the size of one of the columns on the table pair.

   ```
   -- alter main table
   alter table APPLSYS.XYZ_TASKS
     add ( TASK_INFO#1 varchar2(32) );
   exec ad_zd_table.patch('APPLSYS', 'XYZ_TASKS');

   -- alter stage table
   alter table APPLSYS.XYZ_TASKS_SYNC
     add ( TASK_INFO#1 varchar2(32) );
   exec ad_zd_table.patch('APPLSYS', 'XYZ_TASKS_SYNC');
   ```

   As usual, you must create a forward crossedition trigger to populate the new column. Since the FCET logic must be exactly the same for both the main table and stage table, it will make sense to be a little bit clever and develop a single parameterized script that can create the identical trigger on both tables.

   ```
   REM ---- Create FCET ----
   REM dbdrv: sql ~PROD ~PATH ~FILE \
   REM dbdrv:   none none none sqlplus &phase=ccet \
   REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd XYZ_TASKS
   REM dbdrv: sql ~PROD ~PATH ~FILE \
   REM dbdrv:   none none none sqlplus &phase=ccet \
   REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd XYZ_TASKS_SYNC
   REM ---- Apply FCET ----
   REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
   REM dbdrv:   none none none sqlplus &phase=acet \
   REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet XYZ_TASKS_F1
   REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
   REM dbdrv:   none none none sqlplus &phase=acet \
   REM dbdrv:   checkfile:~PROD:~PATH:~FILE:sfcet XYZ_TASKS_SYNC_F1

   REM Copyright (c) 2013 Oracle Corporation, All Rights Reserved
   ```

```
REM $Header$
REM XYZ_TASKS_X1.sql
REM   Update XYZ_TASKS and XYZ_TASKS_SYNC TASK_INFO#1 column

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace trigger &2._F1
  before insert or update on &1..&2
  for each row forward crossedition disable
begin
  :new.task_info#1 := :new.task_info;
end;
/

commit;
exit;
```

The script must be manually applied to the development database to take effect.

```
sqlplus <APPS username> @XYZ_TASKS_X1 APPLSYS XYZ_TASKS
sqlplus <APPS username> @XYZ_TASKS_X1 APPLSYS XYZ_TASKS_SYNC
sqlplus <APPS username> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_TASKS_F1
sqlplus <APPS username> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_TASKS_SYNC_F1
```

Once both main and stage tables are patched, the exchange partition feature will work as expected.

```
delete from xyz_tasks_sync;
insert into xyz_tasks_sync (user_id, task_type, task_info, source)
  values (0, 'OPEN', 'brush kittens', 'SYNCED');
commit;

select * from xyz_tasks;

   USER_ID TASK_TYP TASK_INFO                       SOURCE
---------- -------- ------------------------------ --------
         0 OPEN     take out trash                 LOCAL
         0 OPEN     pull weeds                     SYNCED

alter table applsys.xyz_tasks
  exchange partition synced with table applsys.xyz_tasks_sync
  including indexes without validation;

select * from xyz_tasks;

   USER_ID TASK_TYP TASK_INFO                       SOURCE
---------- -------- ------------------------------ --------
         0 OPEN     take out trash                 LOCAL
         0 OPEN     brush kittens                  SYNCED
```

2. Extract revised table definitions from development database.

   For all instances on R12.AD.C.Delta.8 and later, use the following command. Enter the password when prompted. Alternatively, you could pass the password on the command line but this method might be insecure:

   ```
   perl xdfgen.pl <APPS username> XYZ_TASKS
   perl xdfgen.pl <APPS username> XYZ_TASKS_SYNC
   ```

   For all instances on R12.AD.C.Delta.7:

   ```
   perl xdfgen.pl <APPS username>/<APPS password> XYZ_TASKS
   perl xdfgen.pl <APPS username>/<APPS password> XYZ_TASKS_SYNC
   ```

   For single node database instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <APPS username>/<APPS password>@<DB_SID> XYZ_TASKS
   perl xdfgen.pl <APPS username>/<APPS password>@<DB_SID> XYZ_TASKS_SYNC
   ```

   For RAC instances on R12.AD.C.Delta.6 and earlier:

   ```
   perl xdfgen.pl <APPS username>/<APPS password>@<DB_Instance_Name> XYZ_TASKS
   perl xdfgen.pl <APPS username>/<APPS password>@<DB_Instance_Name> XYZ_TASKS_SYNC
   ```

3. Create the patch.

Patch Files:

- fnd/patch/115/xdf/xyz_tasks.xdf
- fnd/patch/115/xdf/xyz_tasks_sync.xdf
- fnd/patch/115/sql/XYZ_TASKS_X1.sql

Manual Apply actions for the File System:

```
cp fnd/patch/115/xdf/* $FND_TOP/patch/115/xdf
cp fnd/patch/115/sql/* $FND_TOP/patch/115/sql
```

Manual Apply actions for the Database for instances on R12.AD.C.Delta.8 and later:

```
xdfcmp.pl <APPLSYS username> $FND_TOP/patch/115/xdf/xyz_tasks.xdf <APPS username>
xdfcmp.pl <APPLSYS username> $FND_TOP/patch/115/xdf/xyz_tasks_sync.xdf <APPS username>
sqlplus <APPS username> @$FND_TOP/patch/115/sql/XYZ_TASKS_X1 APPLSYS XYZ_TASKS
sqlplus <APPS username> @$FND_TOP/patch/115/sql/XYZ_TASKS_X1 APPLSYS XYZ_TASKS_SYNC
sqlplus <APPS username> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_TASKS_F1
sqlplus <APPS username> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_TASKS_SYNC_F1
```

Manual Apply actions for the Database for instances on R12.AD.C.Delta.7 and earlier:

```
xdfcmp.pl <APPLSYS username>/<APPLSYS password>@$TWO_TASK
$FND_TOP/patch/115/xdf/xyz_tasks.xdf <APPS username>/<APPS password>
xdfcmp.pl <APPLSYS username>/<APPLSYS password>@$TWO_TASK
$FND_TOP/patch/115/xdf/xyz_tasks_sync.xdf <APPS username>/<APPS password>
sqlplus <APPS username> @$FND_TOP/patch/115/sql/XYZ_TASKS_X1 APPLSYS XYZ_TASKS
sqlplus <APPS username> @$FND_TOP/patch/115/sql/XYZ_TASKS_X1 APPLSYS XYZ_TASKS_SYNC
sqlplus <APPS username> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_TASKS_F1
sqlplus <APPS username> @$AD_TOP/patch/115/sql/AD_ZD_TABLE_APPLY XYZ_TASKS_SYNC_F1
```

Exchange Partition failure risk during Online Patch Apply

When following the guidelines in this document, you will patch a main table and staging table for partition exchange as a pair so that the table structures remain identical. But while the online patch is actually running there may be a short period of time where one of the tables is has been patched and the other has not. During this time any attempt to run the "exchange partition" command will fail with the following error:

```
ERROR at line 2:   ORA-14096: tables in ALTER TABLE EXCHANGE PARTITION must have the same number of columns
```

When the online patch has finished patching both tables in the pair, the ability to run "exchange partition" will be restored. Although it is unlikely that such an error will happen in production operation, your runtime application will need to handle this temporary failure case in a reasonable way.

For scheduled repeating processing the simplest way to handle "exchange partition" failure is to exit with a helpful error message and expect that the next scheduled run of the processing will succeed. The error message should indicate that the failure is only temporary while the subject tables are being actively altered by Online Patching, and that the processing should succeed the next time it is run.

For user or event triggered processing the best user experience is achieved by enhancing your code to automatically sleep and then retry the "exchange partition" command until it succeeds. If that option is not feasible, then at least the user should receive a clear error message indicating that they must manually retry the action at a later time.

Dynamically-generated staging tables - Not recommended

The recommended procedure to implement partition exchange is to explicitly create and patch the main table and staging table as a pair of developer-managed tables that always have the same structure. Some products have used an alternative approach in the past, which is to dynamically create a staging table when needed, by querying the structure of the main table and generating dynamic DDL to create a corresponding staging table. This dynamic method is no longer recommended under Online Patching.

If a staging table is generated dynamically, and then the main table structure is altered via an online patch, then the staging table structure will no longer match and the exchange partion command will fail. If the staging table is generated after an online patch has altered the main table, then it will contain the correct physical columns, but the logical view generated by AD_ZD_TABLE.UPGRADE will be incorrect for the run edition. Even if that problem is solved, the generated staging table will not have the necessary crossedition triggers to maintain the revised columns. In short, it is extremely difficult to make a dynamically generated staging table approach work correctly when the main table is patched under online patching.

If your product currently uses the dynamic staging table approach, then there are two options:

1. Convert to a static developer-managed staging table approach, as described earlier in this section.
2. Mark the affected processing as incompatible with Online Patching, so that these issues are avoided. Be aware that Online Patching cycles may last many hours or days, so this approach may be unacceptable to users.

Section 1.5: Deploying Custom Application Tier Objects

This section describes general procedures for deploying custom application tier objects. Use these procedures together with any component-specific steps for the component you are customizing, as described in Section 1.6: Component-Specific Steps for Application Tier Objects.

Section 1.5.1: Setting Up a Production Environment

First, you must set up your custom application on your production environment.

1. On the production environment, run the adop prepare phase.
2. If your customizations include custom Java or BC4J code or extensions, apply the following patches to the production environment.

- 17217965:R12.TXK.C (TEMPLATE CHANGE REQUIRED TO UPLOAD THE CLASS FILES RELATED TO CUSTOMIZATIONS)
- 17217772:R12.AD.C (NEED UTILITY TO GENERATE CUSTOMALL.JAR)

3. Connect to the patch edition file system on the production environment. See: Section 1.1.2: Connecting to the Patch Edition.
4. Invoke adsplice from the patch file system to register your custom application. For more information about using adsplice, see Section 1.3.1: Setting Up an Environment for Customizations.
5. Run the adop cutover phase.

Section 1.5.2: Deploying Customizations on a Production Environment

To deploy customizations, perform the following steps:

1. On the production environment, run the adop prepare phase.
2. Connect to the patch edition file system on the production environment.
3. Copy the custom files to the appropriate directory on the patch edition file system. See the component-specific steps in Section 1.6: Component-Specific Steps for Application Tier Objects.
4. If you copied any custom files under the $JAVA_TOP directory, run the adcgnjar utility to generate and sign a JAR file containing these files. When prompted, enter the user name and password of the APPS user. See Section 1.5.3: Running the adcgnjar Utility.
5. If necessary, use the appropriate utility for your product or component to import or upload the custom files to the database. See the component-specific steps in Section 1.6: Component-Specific Steps for Application Tier Objects.
6. Add entries for the custom files to the custom synchronization driver file to ensure that the adop utility synchronizes these files between the run file system and the patch file system the next time you run the prepare phase. See Section 1.5.4: Adding Entries to the Custom Synchronization Driver File.
7. Run the adop cutover phase.

Section 1.5.3: Running the adcgnjar Utility

Use the adcgnjar utility for any custom Java or BC4J code for Oracle Application Framework, Oracle CRM Technology Foundation (JTT), Oracle Web Applications Desktop Integrator (BNE), custom servlets, or other custom Java code. This utility generates and signs a file named customall.jar file containing the custom Java and BC4J code and extensions. The customall.jar file is included in the ebsProductManifest.xml so that the customall.jar will be in the CLASSPATH through the EBS-product shared library.

The adcgnjar utility does not require any parameters on the command line. When prompted, enter the user name and password of the APPS user. The utility performs the following steps:

1. Creates a temporary custom.zip file that contains all the directories under $JAVA_TOP except the `oracle`, `META-INF`, and `policies` directories.
2. Generates and signs the customall.jar file with the contents of the custom.zip file.
3. Deletes the temporary custom.zip file.

If customall.jar is a client custom jar file, then you must also copy or link `$JAVA_TOP/customall.jar` to `$JAVA_TOP/oracle/apps/xxx/jar/customall.jar` so that it is accessible through URL path `/OA_JAVA/oracle/apps/xxx/jar/customall.jar`.

Section 1.5.4: Adding Entries to the Custom Synchronization Driver File

You should add entries for all your custom files to the custom synchronization driver file located at `$APPL_TOP_NE/ad/custom/adop_sync.drv` (`%s_ne_base%/EBSapps/appl/ad/custom/adop_sync.drv`). The adop utility uses this driver file to synchronize files between the run file system and the patch file system.

Add your entries in the section marked by the '#Begin Customization' and '#End Customization' comments.

When adding your entries, follow the syntax of the examples provided in the `%s_adtop%/admin/template/adop_sync_drv.tmp` template file. For example, if you have custom java class files under the `$JAVA_TOP/<Company identifier>/*` directory, and if all the files under this directory need to be synchronized between the patch file system and the run file system, then add the following entry in the custom synchronization driver file:

```
rsync -zr %s_current_base%/EBSapps/comn/java/classes/<Company identifier>/ %s_other_base%/EBSapps/comn/java/classes
```

You can use context variables in the entries you add. The syntax for a context variable is: `%s_sample_var%`. Context variables are substituted with actual context values before the driver is run; anything in angle brackets (< >) must be substituted by the user with an appropriate value for the system.

Any paths you include in your entries should be specified relative to `s_current_base` and `s_other_base`.

Section 1.5.5: Deploying Java Files at Non-Standard Location(s) for Custom Products

Oracle E-Business Suite standards emphasize having java files that pertain to custom applications at

```
$JAVA_TOP/<Company identifier>/* .
```

When users choose a different location (one other than the standard location given above), for their business requirements, they may need to create a custom jar file manually that contains the custom application's java files at the non-standard location(s) and make this custom jar file available for the WebLogic Server to pick up.

When custom java files are placed in one of the Oracle shipped application's directories, for example,

- `$JAVA_TOP/oracle/apps/fnd`
- `$JAVA_TOP/oracle/apps/ad`
- and so on

then, the custom jar file creation is not necessary. Instead, users need to run "Generate product JAR files" from adadmin which will regenerate product jar files for Oracle applications and that will ensure the custom java files, placed in Oracle shipped application's directories, are on board.

When custom java files are placed in any other non-standard locations, for example,

- `$JAVA_TOP/oracle/<cust_prod>`
- `$JAVA_TOP/oracle/java/<cust_prod>`
- and so on

then the custom jar file must be created manually and it must be made available for WebLogic to pick up. Detailed steps for this procedure are given below.

Creating a custom jar file and making it available:

1. Create a temporary `custom.zip` file which contains all the custom application's directories/files at the non-standard location. The commands are:
    1. `cd $JAVA_TOP`
    2. `zip -r customprod.zip <directory list>` where the `<directory list>` is the list of all the directory paths, relative to $JAVA_TOP, for custom application's java files at the non-standard location.
2. Generate and sign the customprod.jar file. Command: `adjava oracle.apps.ad.jri.adjmx -areas $JAVA_TOP/customprod.zip -outputFile $JAVA_TOP/customprod.jar -jar $CONTEXT_NAME 1 CUST jarsigner -storePass <KeyStore Password> -keyPass <Key Password>`
3. Delete the temporary `customprod.zip` file. Command: `rm $JAVA_TOP/customprod.zip`
4. Follow the steps below for your installation:
    - For installations below R12.TXK.C.DELTA.6 ONLY: Follow the steps below to make the custom jar file available for WebLogic Server:
        1. Back up the existing `<FND_TOP>/admin/template/ebsProductManifest_xml.tmp`
        2. Modify `<FND_TOP>/admin/template/ebsProductManifest_xml.tmp` to add the entry below for `customprod.jar` (after `customall.jar`):

            ```
            <library>customprod.jar</library>
            ```

        3. Run AutoConfig
        4. Bounce the middle-tier services
        5. **NOTE:** These changes will be lost if `ebsProductManifest_xml.tmp` is patched in future; these changes will need to be done again.
    - For installations with R12.TXK.C.DELTA.6 and above:
        1. Create a directory `<FND_TOP>/admin/template/custom`
        2. Copy the existing `<FND_TOP>/admin/template/ebsProductManifest_xml.tmp` to `<FND_TOP>/admin/template/custom`
        3. Modify `<FND_TOP>/admin/template/custom/ebsProductManifest_xml.tmp` to add the entry below for `customprod.jar` (after `customall.jar`):

            ```
            <library>customprod.jar</library>
            ```

        4. Run AutoConfig
        5. Bounce the middle-tier services
        6. **NOTE:** These changes must be redone if `ebsProductManifest_xml.tmp` is patched in future.

5. In order to synchronize the changes (during the next prepare phase) between both the file systems fs1 and fs2, follow the steps below. Note that the custom synchronization driver file, located at `<APPL_TOP_NE>/ad/custom/adop_sync.drv`, should be used in these steps. This file has the required documentation on how to put an entry in for a file that needs to be synchronized between the two file systems.

    1. If there are custom java class files under the `<JAVA_TOP>/oracle/<cust_prod>/*` directory and if the files under this directory needs to be synchronized between fs1 and fs2 , then put the following entry in the custom synchronization driver file as below:

    ```
    cp -r %s_current_base%/EBSapps/comn/java/classes/oracle/<cust_prod>
    %s_other_base%/EBSapps/comn/java/classes/oracle
    ```

    2. To copy the custom jar file, add the following entry:

    ```
    cp %s_current_base%/EBSapps/comn/java/classes/customprod.jar
    %s_other_base%/EBSapps/comn/java/classes
    ```

    3. To synchronize the custom changes done to the template, add the entry below:

    ```
    cp %s_current_base%/EBSapps/appl/fnd/12.0.0/admin/template/ebsProductManifest_xml.tmp
    %s_other_base%/EBSapps/appl/fnd/12.0.0/admin/template
    ```

    4. After changes are synchronized, ensure AutoConfig is run for the latest template changes to take effect.

Section 1.6: Component-Specific Steps for Application Tier Objects

This section lists specific steps required for customizations to particular components or products.

Section 1.6.1: Concurrent Programs

There are a variety of possible customizations for concurrent programs. Examples include:

- Custom concurrent programs to meet specific business requirements.
- Customizations to the Oracle Forms-based Submit Requests window. For example, you can give the Submit Requests window a different title, and define the form so that it allows users to select only those reports or concurrent programs belonging to a request group to which you have assigned a code.
- Customizing the Submit Requests page in Oracle Application Framework. For example, you can customize this page to submit a single request or group of requests, such as a Generate Payroll report.

Section 1.6.1.2: Types of files and their locations

Customizations to the user interface in Oracle Forms or Oracle Application Framework are subject to the customization rules for those interfaces. For program executables, the file locations are as listed below.

- SQ*Plus and PL/SQL : `$<PROD>_TOP/$APPLSQL` or `$<PROD>_TOP/sql`
- PL/SQL stored procedures : Stored in the database
- Oracle Reports : `$<PROD>_TOP/reports/<LANG>`
- SQL*Loader : `$<PROD>_TOP/$APPLBIN`
- C : `$<PROD>_TOP/bin`
- Perl : `$<PROD>_TOP/bin`
- Java File : `$CLASSPATH`

In addition, the file 'afcpprog.lct' is used to upload metadata into the database.

Section 1.6.1.3: Developing Concurrent Program Files

Follow the steps in Section 1.3: Developing Customizations to develop your custom files.

- To develop PL/SQL concurrent programs, use a SQL editor connected to the run edition environment.
- To develop Java concurrent programs, use a Java editor. Ensure that your environment has been set up according to the instructions in Section 1.3.1: Setting Up an Environment for Customizations. Then copy the Java class files to the `$JAVA_TOP/<Company identifier>/*` directory.
- Log in to the Oracle E-Business Suite user interface, navigate to System Administrator > Concurrent > Program, and define the metadata for the concurrent programs.
- Source the run edition environment and download the concurrent program metadata to an LDT file by invoking the Generic Loader (FNDLOAD) utility from the run edition environment with the afcpprog.lct configuration file. See: Using Loaders, Oracle E-Business Suite Setup Guide. For example:

  ```
  FNDLOAD <apps_username>/<apps_password>@<service name> 0 Y DOWNLOAD $FND_TOP/patch/115/import/afcpprog.lct <ldt filename>.ldt PROGRAM APPLICATION_SHORT_NAME=<CUSTOM_TOP>
  ```

  Save a local copy of the LDT file.
- Save local copies of your .pls files and Java class files.

Section 1.6.1.4: Deploying Concurrent Program Files

Follow the steps in Section 1.5: Deploying Custom Application Tier Objects to deploy your custom files.

- Copy your custom files to the following locations:
    - Copy the *.ldt files to the `$<CUSTOM>_TOP/patch/115/import/<LANG>/` folder.
    - Copy the *.pls files to the `$<CUSTOM>_TOP/patch/115/sql` folder.
    - Copy the Java class files to the `$JAVA_TOP/<Company identifier>/*` folder.
- Invoke the the Generic Loader (FNDLOAD) utility with the afcpprog.lct configuration file from the patch edition environment to upload the concurrent program metadata. See: Using Loaders, *Oracle E-Business Suite Setup Guide*.
- Compile any custom PL/SQL objects in the database from the files saved in the `$<CUSTOM>_TOP/patch/115/sql` folder on the patch file system.

Section 1.6.2: Forms

Custom forms are defined as those forms created by the customer or shipped Oracle E-Business Suite forms modified by the customer. Forms personalizations are defined as customer-created metadata that is used at runtime to control the look and/or behavior of a form at runtime. This metadata is created using the Personalizations form (FNDCUSTM.fmb). The personalizations metadata can be downloaded into an LDT file using FNDLOAD and affrmcus.lct. The metadata LDT file can be uploaded using the corresponding LCT file.

Types of Files

- Forms (.fmb and .fmx files)
- Forms Libraries (.pll and .plx files)
- Forms personalizations (controlled by metadata LCT/LDT files)

All the form .fmb files are staged under `$AU_TOP/forms/<LANG>`, for example, `$AU_TOP/forms/US`. The compiled forms (.fmx files) are staged under `$PROD_TOP/forms/<LANG>`, for example, `$FND_TOP/forms/US`.

In the case of custom forms created by a customer, the .fmb files are staged under `$AU_TOP/forms/US`. The compiled forms (fmx files) are staged under `$<CUSTOM>_TOP/forms/US`.

Deploying Forms Files

In addition to these instructions, follow the steps in Section 1.5.4: Adding Entries to the Custom Synchronization Driver File.

Forms (.fmb and .fmx files)

The general instructions for compiling forms in Release 12.2.3 and later are as follows:

1. Source the patch edition environment.

2. Stage the fmb under `$AU_TOP/forms/US`. Verify the $FORMS_PATH environment variable. FORMS_PATH must contain `$AU_TOP/resource` and `$AU_TOP/forms/<LANG>` (if for US this would be `$AU_TOP/forms/US`). If these directories are not set under $FORMS_PATH, set FORMS_PATH accordingly.

3. Compile the .fmb file as follows using MYCUSTOM.fmb as an example:

```
cd $AU_TOP/forms/US
frmcmp_batch MYCUSTOM.fmb <apps_username>/<apps_password>
output_file=$CUSTOM_TOP/forms/US/MYCUSTOM.fmx compile_all=special
```

Here is an example using a version FNDSCAUS.fmb that a customer has modified (customized).

```
cd $AU_TOP/forms/US
frmcmp_batch FNDSCAUS.fmb <apps_username>/<apps_password>
output_file=$FND_TOP/forms/US/MYCUSTOM.fmx compile_all=special
```

For customer-created forms, the `output_file` should point to `$<CUSTOM>_TOP/forms/<LANG>`.

For customer-modified product Oracle E-Business Suite forms, the output_file should point to `$PROD_TOP/forms/<LANG>`.

Any custom forms deployed to the patch edition will become available to the running application after running the cutover phase. Remember to ensure synchronization commands for the custom forms files are included in the custom synchronization driver.

Forms Libraries (.pll and .plx files)

All .pll and .plx files are staged under $AU_TOP/resource. Customer-created .pll and .plx files also reside under $AU_TOP/resource. For any changes made to CUSTOM.pll, as well as customer-created .pll files, the following steps need to be done to deploy them in Release 12.2.

1. Source the patch edition environment.

2. Stage the file under $AU_TOP/resource. Verify the $FORMS_PATH environment variable. FORMS_PATH must contain $AU_TOP/resource directory. If the $FORMS_PATH is not correct, set FORMS_PATH accordingly.

3. Compile the .pll as follows using CUSTOM.pll as an example:

```
cd $AU_TOP/resource
frmcmp_batch CUSTOM.pll <apps_username>/<apps_password> module_type=library
compile_all=special
```

Note that the .plx file created from the above command will be located under `$AU_TOP/resource`.

Stage and compile the .pll files as directed above. After this step is completed, bring the patch edition file system online by running the adop cutover phase. Remember to make sure that the synchronization commands for custom files are defined in the custom synchronization driver. This procedure should keep the two Release 12.2 file systems synchronized with regards to the .pll and .plx files.

Form Personalizations (controlled by metadata LCT/LDT files)

For any form personalizations, you must download the metadata you have created into an .ldt file using the affrmcus.lct and FNDLOAD.

Because affrmcus.lct has PREPARE and TABLE statements, FNDLOAD should handle propagation of the metadata to the patch edition file system.

Folders Configuration Customizations (controlled by metadata LCT/LDT files)

For folders, use the Folder file for FNDLOAD. These files are documented in the *Oracle E-Business Suite Setup Guide* in the "Using Loaders" chapter. Note that fndfold.lct does NOT have PREPARE statements because fndfold.lct is for customer use only and there are no shipped .ldt files that call fndfold.lct.

To download all folders:

```
FNDLOAD username@database 0 Y DOWNLOAD $FND_TOP/patch/115/import/fndfold.lct <name of file>.ldt FND_FOLDERS
```

To upload folders:

```
FNDLOAD username@database 0 Y UPLOAD $FND_TOP/patch/115/import/fndfold.lct <name of file>.ldt
```

You will be prompted for the password with the above commands.

Section 1.6.3: Oracle Application Framework

Customers who have installed or upgraded to Release 12.2.4 or later should review *Oracle E-Business Suite Release 12.2 Upgrade Considerations for OAF-based Applications and Oracle CRM Technology Foundation*, My Oracle Support Document 1927975.1.

Other sections in this document may use the terms personalization, customization, and extension interchangeably; however, for Oracle Application Framework please note that these terms mean three different things. See: Deploying Customer Extensions, *Oracle Application Framework Developer's Guide* (available from My Oracle Support Document 1315485.1) and Personalizing Your Pages and Portlets,*Oracle Application Framework Personalization Guide*.

Section 1.6.3.1: OA Framework Personalizations: Types of Files

Once you create a personalization, OA Framework inserts the metadata into the relevant Oracle MDS repository tables. You can export the MDS metadata for the personalization in the form of an XML or XLIFF file using the XMLExporter or XLIFF Extractor utilities, respectively.

You can run the XMLImporter or XLIFF Importer utility from the command line or from the Functional Administrator responsibility to import the personalization in the XML/XLIFF file into another MDS repository and deploy the personalization immediately. See: Deploying Personalizations,Oracle Application Framework Personalization Guide and Translating Personalizations, Oracle Application Framework Personalization Guide. If you have Oracle JDeveloper OA Extension, you may alternatively use the import.bat file or the import shell script that is packaged with the JDeveloper IDE, located in the jdevbin\oaext\bin directory of the JDeveloper install area. The batch file and shell script each set up the classpath, path and environment for you. Just typing import without any parameters will give help about its usage.

To deploy a personalization via a patch, follow the instructions in Section 1.6.3.2: Deploying OA Framework Personalizations Via a Patch below. Note that in this case, the personalization does not become available until after the cutover.

Section 1.6.3.2: Deploying OA Framework Personalizations Via a Patch

To deploy your personalization via a patch:

On the source Oracle E-Business Suite instance where you create your personalization:

Source the run edition environment.

Export the personalized MDS metadata using the XMLExporter or XLIFF Extractor utility. Refer to the section Deploying Personalizations and the section Translating Personalizations of the *Oracle Application Framework Personalization Guide*. If you have Oracle JDeveloper OA Extension, you may alternatively use the export.bat / xliffextract.bat files or the export / xliffextract shell scripts that are packaged with the JDeveloper IDE, located in the jdevbin\oaext\bin directory of the JDeveloper install area. The batch files and shell scripts each set up the classpath, path and environment for you. Just typing export or xliffextract without any parameters will give help about its usage.

On the target Oracle E-Business Suite instance:

Perform the setup steps described in Section 1.5.1: Setting Up a Production Environment.

Run the adop prepare phase.

Source the patch edition environment.

Copy the exported XML/XLIFF file from step 1B to the $<CUSTOM>_TOP/mds directory, where $<CUSTOM>_TOP refers to your custom product top.

Invoke the XMLImporter or XLIFFImporter utility on the Patch file system to load the contents of the XML/XLIFF file to the MDS repository.

Note: Running the XLIFFImporter manually will make the personalization available immediately.

Follow the instructions in Section 1.5.4: Adding Entries to the Custom Synchronization Driver File to add an entry of the XML file into the custom synchronization driver file. This ensures that the custom files are synchronized between the Run and Patch file systems the next time you run the adop prepare phase.

Run the adop cutover phase.

Section 1.6.3.3: OA Framework Business Logic Extensions: Types of Files

- OA Extension Controller Java files
- BC4J XML files
- BC4J Substitutions
- JRAD XML files

Section 1.6.3.4: Developing OA Framework Business Logic Extensions

To develop business logic extensions:

1. Use Oracle JDeveloper with OA Extensions to create your business logic extensions in the Source system. See: Extending OA Framework Applications, Oracle Application Framework Developer's Guide, available from .

2. Follow the initial setup instructions described in Section 1.3: Developing Customizations.

3. Perform the following steps in the Run file system:

    1. Source the run edition environment.

    2. For OA Extension Controller Java class file changes and BC4J XML file changes, copy the .class and .xml files to `$JAVA_TOP/<Company identifier>/*`.

3. Run the adcgnjar utility, as described in Section 1.5.3: Running the adcgnjar Utility.
4. For BC4J substitutions and JRAD XML file changes, copy the JPX and XML files to the `$<CUSTOM>_TOP/mds` directory, where `$<CUSTOM>_TOP` refers to your custom product top.
5. Invoke the JPXImporter and XMLImporter utilities in the Run file system to load the contents of the JPX and XML files to the MDS repository. For information on these utilities, refer to the Oracle Application Framework Developer's Guide, available from My Oracle Support Document 1315485.1. If you have Oracle JDeveloper OA Extension, you may alternatively use the jpximport.bat file, import.bat or the import shell script that is packaged with the JDeveloper IDE, located in the `jdevbin\oaext\bin` directory of the JDeveloper install area. The batch files and shell script each set up the classpath, path and environment for you. Just typing jpximport or import without any parameters will give help about its usage.
6. Follow the instructions in Section 1.5.4: Adding Entries to the Custom Synchronization Driver File to add an entry of the XML file into the custom synchronization driver file. This ensures that the custom files are synchronized between the Run and Patch file systems the next time you run the adop prepare phase.

Section 1.6.3.5: Deploying OA Framework Business Logic Extensions

To deploy business logic extensions:

1. Follow the initial setup instructions in Section 1.5.1: Setting Up a Production Environment.
2. Run the adop prepare phase.
3. Source the patch edition environment.
4. For OA Extension Controller Java class file changes and BC4J XML file changes, copy the .class and .xml files to `$JAVA_TOP/<Company identifier>/*`.
5. Run the adcgnjar utility, as described in Section 1.5.3: Running the adcgnjar Utility.
6. For BC4J substitutions and JRAD XML file changes, copy the JPX and XML files to the `$<CUSTOM>_TOP/mds` directory, where `$<CUSTOM>_TOP` refers to your custom product top.
7. Invoke the JPXImporter and XMLImporter utilities in the Patch file system to load the contents of the JPX and XML files to the MDS repository. For information on these utilities, refer to the Oracle Application Framework Developer's Guide (available from My Oracle Support Document 1315485.1). If you have Oracle JDeveloper OA Extension, you may alternatively use the jpximport.bat file, import.bat or the import shell script that is packaged with the JDeveloper IDE, located in the `jdevbin\oaext\bin` directory of the JDeveloper install area. The batch files and shell script each set up the classpath, path and environment for you. Just typing jpximport or import without any parameters will give help about its usage.
8. Follow the instructions in Section 1.5.4: Adding Entries to the Custom Synchronization Driver File to add an entry of the XML file into the custom synchronization driver file. This ensures that the custom files are synchronized between the Run and Patch file systems the next time you run the adop prepare phase.
9. Run the adop cutover phase.

Section 1.6.4: Oracle CRM Technology Foundation

Custom files for Oracle CRM Technology Foundation (JTT) can be of the following types:

- js
- jsp
- css
- xss
- xsl
- htm
- html
- and so on

Section 1.6.4.2: Developing JTT Customizations

Follow the steps in Section 1.3: Developing Customizations.

To test customizations in a development environment, perform the following steps:

1. Copy the newly created js, jsp, css, xss, xsl, htm, html and other files to the $OA_HTML/* directory.
2. To render custom jsp files in the browser, ensure that the profile option "Allow Unrestricted JSP Access" is set to "Yes".
3. Develop Java class files using any Java editor tool and copy the Java class files to the `$JAVA_TOP/<Company identifier>/*` directory.

You should also save local copies of all your custom files. From the development environment, source the run edition file system and then save the files.

Section 1.6.4.3: Deploying JTT Customizations

1. First, ensure you have performed the setup steps in Section 1.5.1: Setting Up a Production Environment.
2. Log into the Oracle E-Business Suite user interface running from the run edition environment and ensure that the profile option "Allow Unrestricted JSP Access" is set to "Yes".
3. Then follow the steps in Section 1.5.2: Deploying Customizations on a Production Environment.
    1. Copy any custom Java class files to the `$JAVA_TOP/<Company identifier>/*` directory.
    2. Copy your other custom files, including js, jsp, css, xss, xsl, htm, html, and other files to the `$<CUSTOM>_TOP/html/*` directory and mirror copy the same files to the `$OA_HTML/*` directory.

Section 1.6.5: Oracle Web Applications Desktop Integrator

Use this section for Oracle Web Applications Desktop Integrator customizations.

Section 1.6.5.1: Developing Oracle Web Applications Desktop Integrator Customizations

You can create custom integrators using Oracle E-Business Suite Desktop Integration Framework. You can create custom integrators for seeded Oracle E-Business Suite applications or for your own custom application. All objects created in the integrator metadata are marked with the application ID and are uniquely identified by a combination of the application ID and the object's internal code name. See: *Oracle E-Business Suite Desktop Integration Framework Developer's Guide*.

Follow the steps in Section 1.3: Developing Customizations to develop your custom files.

- Use Oracle E-Business Suite Desktop Integration Framework user interface to define your custom integrator metadata.
- If your integrator uses any custom PL/SQL packages and functions, you can develop these using a SQL editor while connected to the run edition environment.
- If your integrator uses any custom Java classes, then develop these in a Java editor and copy the Java class files to the `$OA_HTML/WEB-INF/lib/*` directory and to the `$JAVA_TOP/<Company identifier>/*` directory.
- Source the run edition environment and download the integrator metadata to an LDT file by invoking the Generic Loader (FNDLOAD) utility from the run edition environment with the bneintegrator.lct configuration file. Save a local copy of the LDT file. See: Loading Integrator Definitions, *Oracle E-Business Suite Desktop Integration Framework Developer's Guide*.
- Save local copies of your .pls files and Java class files.

Section 1.6.5.2: Deploying Oracle Web Applications Desktop Integrator Customizations

Follow the steps in Section 1.5: Deploying Custom Application Tier Objects to deploy your custom files.

- Copy your custom files to the following locations:
    - Copy the *.ldt files to the `$<CUSTOM>_TOP/patch/115/import/<LANG>/` folder.
    - Copy the *.pls files to the `$<CUSTOM>_TOP/patch/115/sql` folder.
    - Copy the Java class files to the `$JAVA_TOP/<Company identifier>/*` folder.
- Invoke the Generic Loader (FNDLOAD) utility with the `bneintegrator.lct` configuration file from the patch edition environment to upload the integrator metadata. See: Loading Integrator Definitions, *Oracle E-Business Suite Desktop Integration Framework Developer's Guide*.
- Compile any custom PL/SQL objects in the database from the files saved in the `$<CUSTOM>_TOP/patch/115/sql` folder on the patch file system.

Note: Note: Oracle does not support custom integrators created through PL/SQL APIs rather than through Oracle E-Business Suite Desktop Integration Framework. However, if you have a custom integrator created through APIs, you can follow the same general steps listed in this document to deploy it in Release 12.2.

Section 1.6.6: Oracle Workflow

In Oracle Workflow, you can create the following types of customizations:

- Workflow processes
    - Customizations to seeded workflow processes
    - New custom workflow processes

- Business events and event subscriptions
    - Customizations to seeded business events and event subscriptions
    - New custom business events and event subscriptions
- Approvals data services configuration metadata
    - New custom message configurations
    - New custom approval type definitions

The following sections describe how to develop and deploy each of these types of customizations. For more information, see the *Oracle Workflow Developer's Guide*. In particular, see the Customization Guidelines section in Appendix C.

Section 1.6.6.1: Customizing Workflow Processes

Use this section for Oracle Workflow customizations.

Developing Workflow Process Customizations

1. For workflows seeded by Oracle E-Business Suite products, check your product-specific documentation to determine whether any customizations are supported or required. If so, you can use the Oracle Workflow Builder client tool to edit the workflow definition. You can access the seeded workflow definition by either of the following methods:
    - Use Oracle Workflow Builder to open a copy of the *.wft file from the run file system of your development environment. The .wft files for a product are usually located in the `$PROD_TOP/patch/115/import/<LANG>/` directory.
    - Use Oracle Workflow Builder to connect to the run edition of your development database and open the workflow definition stored in the database.

    You can also create your own new custom workflow processes using Oracle Workflow Builder.
2. For both customizations of seeded workflows and new custom workflows, save your workflow definition to a *.wft file in the `$<CUSTOM>_TOP/patch/115/import/<LANG>/` folder, where `$<CUSTOM>_TOP` refers to your custom product top.
    - You can use Oracle Workflow Builder to save the workflow definition as a *.wft file.
    - If you have saved the workflow definition to your development database, you can also use the Workflow Definitions Loader concurrent program (WFLOAD) to download the workflow definition from the database to a *.wft file. Ensure that you connect to the run edition of your development database when running the Workflow Definitions Loader. See: Using the Workflow Definitions Loader, *Oracle Workflow Administrator's Guide*.
3. You can use either of the following methods to save the workflow definition to your development database for testing.
    - Use Oracle Workflow Builder to connect to the run edition of your development database and save the workflow definition to the database.
    - If you have saved the workflow definition to a *.wft file, you can also use the Workflow Definitions Loader concurrent program to upload the workflow definition to the database. Ensure that you connect to the run edition of your development database when running the Workflow Definitions Loader. See: Using the Workflow Definitions Loader, *Oracle Workflow Administrator's Guide*.
4. If you reference a custom PL/SQL function in a function activity in your workflow, then save that PL/SQL package and function definition locally in a *.pls file.
5. If you reference a custom event in an event activity, then download the metadata for that event from your development database to a *.wfx file and copy that file to the `$<CUSTOM>_TOP/patch/115/xml/<LANG>` folder, where `$<CUSTOM>_TOP` refers to your custom product top. First source the run file system of your development environment and then use the Workflow XML Loader utility (WFXLoad) to download the *.wfx file. See: Using the Workflow XML Loader, *Oracle Workflow Administrator's Guide*.

Deploying Workflow Process Customizations

Follow the general steps in Section 1.5: Deploying Custom Application Tier Objects.

- Copy your custom files to the following locations on the patch file system of your production database:
    - *.wft files - `$<CUSTOM>_TOP/patch/115/import/<LANG>/` folder
    - *.wfx files - `$<CUSTOM>_TOP/patch/115/xml/<LANG>/` folder
    - *.pls files - `$<CUSTOM>_TOP/patch/115/sql` folder
- Upload your custom files to the production database as follows:
    - Use the Workflow Definitions Loader concurrent program to upload the workflow definitions from the .*wft files on the patch file system to the database.
    - Use the Workflow XML Loader utility to upload any event and subscription metadata from the *.wfx files on the patch file system to the database.
    - Compile any custom PL/SQL objects in the database from the files saved in the `$<CUSTOM>_TOP/patch/115/sql` folder on the patch file system.

Section 1.6.6.2: Customizing Business Events and Subscriptions

This section describes customization of business events and subscriptions in Oracle Workflow..

Developing Event and Subscription Customizations

1. For business events or subscriptions seeded by Oracle E-Business Suite products, check your product-specific documentation to determine whether any customizations are supported. If so, you can use the Oracle Workflow Event Manager user interface in Oracle E-Business Suite to edit the event or subscription definition.

    You can also create your own new custom events and subscriptions using the Event Manager.

    Note: Before you create a custom event or subscription owned by a custom product, ensure that your custom product is registered and licensed according to the instructions in Section 1.3: Developing Customizations.
2. If any of your events have a custom PL/SQL generate function, or if any of your subscriptions have a custom PL/SQL rule function, then save that PL/SQL package and function definition locally in a *.pls file.
3. If any of your events have a custom Java generate function, or if any of your subscriptions have a custom Java rule function, then copy the Java class file to `$OA_HTML/WEB-INF/lib/*` and to `$JAVA_TOP/<Company identifier>/*`, and also save the Java class file locally.
4. Download the event and subscription metadata from your development database to a *.wfx file. First source the run file system of your development environment and then use the Workflow XML Loader utility (WFXLoad) to download the *.wfx file. See: Using the Workflow XML Loader, *Oracle Workflow Administrator's Guide*.

Deploying Event and Subscription Customizations

Follow the general steps in Section 1.5: Deploying Custom Application Tier Objects.

- Copy your custom files to the following locations on the patch file system of your production database:
    - *.wfx files - `$<CUSTOM>_TOP/patch/115/xml/<LANG>/` folder
    - *.pls files - `$<CUSTOM>_TOP/patch/115/sql folder`
    - Java files - `$JAVA_TOP/<Company identifier>/*` folder
- Upload your custom files to the production database as follows:
    - Use the Workflow XML Loader utility to upload the event and subscription metadata from the *.wfx files on the patch file system to the database.
    - Compile any custom PL/SQL objects in the database from the files saved in the `$<CUSTOM>_TOP/patch/115/sql` folder on the patch file system.
    - If you have any custom Java files under the $JAVA_TOP folder, ensure that you run the adcgnjar utility as described in Section 1.5.3: Running the adcgnjar Utility.

Section 1.6.6.3: Customizing Approvals Data Services Configuration Metadata

Use this section for developing and deploying custom approvals data services configuration metadata.

Developing Custom Approvals Data Services Configuration Metadata

If you want to expose approval notifications from custom workflows in the Oracle Mobile Approvals for Oracle E-Business Suite app, you can define custom configuration metadata so that the approvals data services can communicate your notification data to the app. You can create custom message configurations and approval type definitions using the approvals data services configuration tool available in the Oracle Workflow administrator UI.

Note: Customizations to message configurations and approval type definitions seeded by Oracle E-Business Suite are not supported.

Follow the steps in Section 1.3: Developing Customizations to develop your custom files.

- Use the approvals data services configuration tool to define your custom message configurations and approval type definitions. See: Configuring Messages for Oracle Mobile Approvals for Oracle E-Business Suite, *Oracle Workflow Developer's Guide*.
- Source the run edition environment and download the message configurations and approval type definitions to LDT files by invoking the Generic Loader (FNDLOAD) utility from the run edition environment with the wfwlsvcs.lct configuration file. Save a local copy of each LDT file. See: Loading Approvals Data Services Configuration Metadata, *Oracle Workflow Developer's Guide*.

Note: If you need to make any changes in your custom workflows while preparing to configure your approvals data services metadata, then follow the instructions in Section 1.6.6.1: Customizing Workflow Processes. If you need to make any changes in your Oracle Application Framework code, then follow the instructions in Section 1.6.3.4: Developing OA Framework Business Logic Extensions and Section 1.6.3.5: Deploying OA

Framework Business Logic Extensions.

Deploying Custom Approvals Data Services Configuration Metadata

Follow the steps in Section 1.5: Deploying Custom Application Tier Objects to deploy your custom files.

- Copy your custom *.ldt files to the `$<CUSTOM>_TOP/patch/115/import/<LANG>/` folder.
- Invoke the Generic Loader (FNDLOAD) utility with the wfwlsvcs.lct configuration file from the patch edition environment to upload the message configurations and approval type definitions. See: Loading Approvals Data Services Configuration Metadata, *Oracle Workflow Developer's Guide*.

Section 1.6.7: Oracle XML Gateway

Use this section for Oracle XML Gateway customizations.

Section 1.6.7.1: Developing Oracle XML Gateway Customizations

Oracle XML Gateway allows you to develop XML Gateway Maps and Document Type Definitions (DTDs) to meet your business needs. For more information about using and customizing Oracle XML Gateway, see: *Oracle XML Gateway User's Guide*.

In addition to the steps in Section 1.3: Developing Customizations, use the following guidelines to develop customizations for Oracle XML Gateway:

- Use the XML Gateway Message Designer and configure it to connect to the run edition, which is the default.
- Use XML Gateway Message Designer to create an XML Gateway Map. Since XML Gateway Message Designer cannot directly upload XML Gateway Maps, save the custom Map definition (`*.xgm`) in a local copy.
- If the XML Gateway Map is created with any custom DTD, then use any XML editor to develop the custom DTD and save a local copy (`*.dtd`).
- If the XML Gateway Map has any Action defined for custom XSL Transformation, then use any XML editor to develop the XSLT and save a local copy (`*.xsl`).
- If the XML Gateway Map has any Action defined to run any custom procedure, then use any SQL development tool to create packages and procedures and save a local copy (`.pls`).

Section 1.6.7.2: Deploying Oracle XML Gateway Customizations

Use the following guidelines to deploy your customizations for Oracle XML Gateway on a target production environment:

- Follow the general steps in Section 1.5: Deploying Custom Application Tier Objects.
- Copy your saved custom files to the following locations on the patch file system of your production database:
  - Copy the `.xgm` file to `$<CUSTOM>_TOP/patch/115/xml/<LANG>/`
  - Copy the `.dtd` and `.xsl` files to `$<CUSTOM>_TOP/patch/115/xml/`
  - Copy the `.pls` file to `$<CUSTOM>_TOP/patch/115/sql/`
- On the target instance, load the custom files using the following loader utilities:
  - Invoke the LoadDTDToClob utility on the patch file system to load the contents of the document type definition `*.dtd` file.
  - Invoke the LoadXSLTToClob utility on the patch file system to load the contents of the XSL Transformation `*.xsl` file.
  - Invoke the LoadMap utility on the patch file system to load the contents of the XML Gateway Map `*.xgm` file.
- Compile any custom PL/SQL objects in the database from the `*.pls`files saved in the `$<CUSTOM>_TOP/patch/115/sql` folder on the patch file system.

Section 1.6.8 Oracle XML Publisher

Use this section for customizations with Oracle XML Publisher.

Section 1.6.8.1: Developing Oracle XML Publisher Customizations

For Oracle XML Publisher, you can create the following types of custom files.

- Data templates - These files are of type `*.xml` and can include data templates, sample XML, and bursting control files.
- Layout templates - You can create the following types of layout templates:
  - `*.pdf` - PDF template
  - `*.rtf` - RTF template or E-Text template
  - `*.xsl` - XSL template
  - `*.xls` - Microsoft Excel template
- Java class files - You can create custom Java class files if you use Java concurrent programs as the interface to generate Oracle XML Publisher reports.

Follow the steps in Section 1.3: Developing Customizations to develop your custom files.

- Create your custom data templates, sample XML, bursting control files, and layout templates according to the following guides, which are available through the Oracle E-Business Suite online help:
  - *Oracle XML Publisher Administration and Developer's Guide*
  - *Oracle XML Publisher Report Designer's Guide*
- Log in to the Oracle E-Business Suite user interface, navigate to XML Publisher Administration, and upload your data templates and layout templates using the Oracle XML Publisher user interface.
- Write the sample code to generate the Oracle XML Publisher reports using your custom files. For more information, see the *Oracle XML Publisher Report Designer's Guide*.
- If you choose to use Java concurrent programs as the interface to generate Oracle XML Publisher reports, use a Java editor to develop the Java concurrent programs. Ensure that you follow the concurrent program SDK and the Oracle XML Publisher SDK. Then copy the Java class files to the `$JAVA_TOP/<Company identifier>/*` directory.
- Source the run edition environment and download the Oracle XML Publisher metadata to an LDT file by invoking the Generic Loader (FNDLOAD) utility from the run edition environment with the $XDO_TOP/patch/115/import/xdotmpl.lct configuration file. Save a local copy of the LDT file. See: Using Loaders, *Oracle E-Business Suite Setup Guide* and the *Oracle XML Publisher Administration and Developer's Guide*.
- Save local copies of your data templates, sample XML, bursting control files, layout templates, and any Java class files.

Section 1.6.8.2: Deploying Oracle XML Publisher Customizations

Follow the steps in Section 1.5: Deploying Custom Application Tier Objects to deploy your custom files.

- Copy your custom files to the following locations:
  - Copy the *.ldt files to the `$<CUSTOM>_TOP/patch/115/import/<LANG>/` folder.
  - Copy *.xml files, including data templates, sample XML, and bursting control files, to the $<CUSTOM>_TOP/patch/115/publisher/defs folder.
  - Copy all types of layout templates to the `$<CUSTOM>_TOP/patch/115/publisher/templates/<LANG>` folder.
  - Copy the Java class files to the `$JAVA_TOP/<Company identifier>/*` folder.
- Upload your custom files using the following utilities:
  - Invoke the Generic Loader (FNDLOAD) utility with the xdotmpl.lct configuration file from the patch edition environment to upload the Oracle XML Publisher metadata from your *.ldt files. See: Using Loaders, *Oracle E-Business Suite Setup Guide*.
  - Use the XDOLoader utility from the patch edition environment to load the *.xml files, including data templates, sample XML, and bursting control files, as well as all types of layout templates. See: *Oracle XML Publisher Administration and Developer's Guide* in the Oracle E-Business Suite online help.

Section 1.7: Troubleshooting Deployment of Customizations

This section lists troubleshooting tips.

- If after a cutover there is a discrepancy in file versions, the discrepancy may cause the original file not to be replaced.

For more troubleshooting tips about the online patching cycle, see the "Diagnostics and Troubleshooting" section in Chapter 3, Patching Procedures, of the *Oracle E-Business Suite Maintenance Guide*.

Part 2: Database Object Development Standards for Online Patching

Section 2.1: Introduction

This section documents new or modified standards for database object development and online patching in Oracle E-Business Suite Release 12.2. This section only covers standards that are required for Online Patching support; it does not integrate other general development standards. This section is organized by Object Type. For each Object Type we list out the following types of standards:

- Definition Standards: Rules that must be followed concerning the definition of the object.
- Usage Standards: Rules that must be followed when using the object during application runtime.
- Dynamic DDL Standards: Rules that must be followed when generating or altering the object definition during application runtime.
- Online Patching Compliance Standards: Rules that must be followed when delivering the object definition in an online patch. These rules apply to all customizations, patches, and upgrades on top of Oracle E-Business Suite Release 12.2. These rules do NOT apply to patches that make up the Oracle E-Business Suite Release 12.2 upgrade itself.

Full Compliance versus Runtime Compliance

Full compliance with all standards in this section enable a product to support both correct runtime operation and online patching in an Oracle E-Business Suite Release 12.2 environment. Runtime compliance is a subset of the full standards that allows a product operate correctly in the Oracle E-Business Suite Release 12.2 runtime, but without supporting online patching of the product itself. A product that only implements runtime compliance must be patched using downtime patching. Oracle E-Business Suite products are required to implement full compliance, but custom or third-party products have the option to only implement the runtime compliance subset. Runtime compliance standards are marked with a **[minimal]** tag.

Note: These standards only apply to objects and data that are directly managed by application developers. The database may generate internal/hidden tables, indexes, types, and other objects to implement certain higher level features. These development standards do not apply to these generated objects.

Note: Any reference to an explicit schema name in these standards (e.g. "APPS") should be taken as conceptual, and not necessarily the actual schema name. Schema names in Oracle E-Business Suite may have been changed during installation, and so must not be hard-coded. The only exception to this rule is the "APPS_NE" schema which is guaranteed to have this name on all installations.

Note: This document does not attempt to explain Online Patching. Please refer to the *Oracle E-Business Suite Maintenance Guide*, Part No. E22954, for a description of Online Patching concepts and operation.

General Information

The Online Patching Database Compliance Checker Report

This utility reports all violations to the Online Patching development standards for database objects. You must fix any object listed in this report that is part of your custom code. If you do not fix the violations, then you cannot use the online patching infrastructure to patch the objects listed in this report. Check for violations of online patching database standards by running the Online Patching Database Compliance Checker with the following command:

```
sqlplus <APPS username> @$AD_TOP/sql/ADZDDBCC.sql
```

Some of these standards are marked with a code corresponding to the output of the Online Patching Database Compliance Checker. For example, "dbcc:SECTION-17".

Note: For more information on this utility, refer to *Oracle E-Business Suite Developer's Guide* and My Oracle Support Knowledge Document 1531121.1, *Using the Online Patching Readiness Report in Oracle E-Business Suite Release 12.2*. Also refer to the README for the patch referenced in that document.

The Global Standards Compliance Checker (GSCC) Report

For some of these standards, a code corresponding to the Global Standards Compliance Checker (GSCC) script output is given. An example of a code is "GSCC File.Gen.34".

Note: For more information on this utility, refer to My Oracle Support Knowledge Document 1531121.1, *Using the Online Patching Readiness Report in Oracle E-Business Suite Release 12.2*, as well as the README for the patch referenced in that document.

Requirements of Online Patching

The two main requirements of online patching can be stated in two simple rules:

- An online patch must not break the running application
- The running application must not break the online patch **[minimal]**

The basis of Online Patching is an Oracle Database feature known as Edition-Based Redefinition (EBR). This means that the Oracle E-Business Suite installation supports two editions (versions) of the application on a single system. The application runs on a set of files and database objects known as the "Run Edition". An online patch is applied to a copy of the Run Edition known as the "Patch Edition". Database tables, indexes, and other non-editioned objects are shared across both editions.

The requirements for Online Patching imply the following general development standards:

- An online patch must only change editioned objects in the Patch Edition.
- An online patch must not make incompatible changes to non-editioned objects or data used by the running application.
- The running application must replicate dynamic changes to editioned objects to the Patch Edition. **[minimal]**
- The running application must not make changes to non-editioned objects maintained by Online Patching. **[minimal]**

The remaining development standards provide guidance to help ensure you meet the above criteria for each type of database object.

Section 2.2: Editioned Objects

Editioned objects may have different definitions in each database edition. This means that such objects can be easily patched (in the patch edition) without affecting the running application (in the run edition).

The following object types are covered:

- View (Ordinary)
- PL/SQL Package
- User-Defined Type (Editioned)
- PL/SQL Trigger
- Synonym
- Virtual Private Database Policy

Section 2.2.1: Granting Privileges to Editioned Objects

This procedure allows a developer to grant privileges on Oracle E-Business Suite objects without causing object invalidation.

There are occasions where you might want to grant privileges to objects in the APPS schema; for example, you might want to create a database user with read-only privileges. However, due to the way that editioned objects are treated, granting a privilege on an editioned object directly may cause temporary invalidation of any dependent objects.

To avoid such invalidation, use the procedure AD_ZD.GRANT_PRIVS:

```
procedure  GRANT_PRIVS( X_PERMISSIONS in VARCHAR2,
                        X_OBJECT_NAME in  VARCHAR2,
                        X_GRANTEE     in VARCHAR2,
                        X_OPTIONS     in VARCHAR2 default NULL);
end;
```

Parameters:

- X_PERMISSIONS: Permissions to be granted to the grantee. This should be in upper case.
- X_OBJECT_NAME: Object to which the permission(s) apply. This should be in the exact case as defined and must exist in the database. By default, objects will have uppercase names.
- X_GRANTEE: Other schemas and roles receiving the grant.
- X_OPTIONS: Grant options, for example: 'WITH GRANT OPTION'.

Section 2.2.2: View (Ordinary)

A view is a logical representation of one or more tables. It can be thought of as a stored query that derives its data from the tables on which it is based. For more information on views, see: *Oracle Database Administrator's Guide*.

Definition Standards

- A view name must be a non-quoted identifier. (dbcc: SECTION-10)

  Note: Non-quoted identifiers may only use the following characters: A-Z 0-9 _ # $. Quoted identifiers are reserved for use by the technology components.

- It is recommended that a join view define a ROW_ID column mapped to the ROWID of the base table. **[minimal]**

  This is a "best practice" guideline for custom views. The reasoning behind this recommendation is that because you cannot rely on the "ROWID" pseudocolumn being mapped to the correct base table in a join view, it is better to create an explicit "ROW_ID" view column that gets the ROWID of the actual base table for the join view.

  However, this standard is not strictly required for custom views. If you do not need to have access to the base table ROWID then you do not need to expose that column through your custom view. The guideline exists because IF you want to access to the ROWID of the base table of a join view, THEN do not rely on the database to automatically propagate the ROWID pseudocolumn; instead, define an explicit ROW_ID column in the join view definition.

Usage Standards

Do not reference the implicit ROWID pseudo-column of a join view. The availability of implicit ROWID from a join view may change without warning. Instead, use the explicit ROW_ID column that should be present in all join views. **[minimal]**

Dynamic DDL Standards

If the running application creates, replaces or drops a view while a patch edition exists, then the same action must be run in the patch edition. This requirement can be satisfied in either of two ways:

1. Replicate the run edition dynamic DDL in the patch edition.

   Use AD_DDL, which will replicate run edition DDL for views in the patch edition automatically.

   Example:

```
/* Code example:  Create a view using AD_DDL */

declare
  L_APP   varchar2(8);
  L_NAME  varchar2(30);
  L_STMT  varchar2(32000);
begin
  l_app  := 'FND';
  l_name := 'FND_EXAMPLE_V';
  l_stmt :=
    'create or replace view '||l_name||' as '||
    'select user_id, user_name from fnd_user '||
    'where user_id < 10000';

  ad_ddl.do_ddl(
    ad_zd.applsys_schema,  -- applsys schema name
    l_app,                 -- application short name for your product
    ad_ddl.create_view,    -- statement type
    l_stmt,                -- statement text
    l_name                 -- view name
  );
end;
/
```

2. Disable or block application processing that runs dynamic DDL during online patching.

   This approach can only be used if the dependent functionality can be unavailable for days at a time without disrupting normal customer operations.

   See:

   - Section 2.5.1: Example of Disabling Functionality while Online Patching
   - Section 2.5.2: Blocking a Concurrent Program while Online Patching

Online Patching Compliance Standards

Do not install views in non-editioned database schemas such as CTXSYS, SYSTEM, SYS, or PUBLIC.

Section 2.2.3: PL/SQL Package

For information on PL/SQL packages, see: *Oracle Database PL/SQL Language Reference.*

Definition Standards

The package name must be a non-quoted identifier. (dbcc: SECTION-10)

Note: Non-quoted identifiers may only use the following characters: A-Z 0-9 _ # $. Quoted identifiers are reserved for use by the technology components.

Usage Standards

No special considerations.

Dynamic DDL Standards

If the running application creates, replaces, or drops PL/SQL while a patch edition exists, then the same action must be run in the patch edition. This requirement can be satisfied in either of two ways:

1. Replicate the run edition dynamic DDL in the patch edition.

   Use AD_DDL, which will replicate the run edition DDL for PL/SQL in the patch edition automatically.

   For example:

```
-- Example of using AD_DDL to create a PL/SQL package

DECLARE
  l_applsys_schema varchar2(30);
BEGIN
  -- Get applsys schema name
  SELECT user
    INTO l_applsys_schema
    FROM dual;

  -- Build and create Package Spec
  ad_ddl.build_package('create package FND_EX_TEST as', 1);
  ad_ddl.build_package('  procedure test;', 2);
  ad_ddl.build_package('end;', 3);
  ad_ddl.create_package(l_applsys_schema, 'FND', 'FND_EX_TEST', 'FALSE', 1, 3);

  -- Build and create Package Body
  ad_ddl.build_package('create package body FND_EX_TEST as', 1);
  ad_ddl.build_package('  procedure test is', 2);
  ad_ddl.build_package('  begin null; end;', 3);
  ad_ddl.build_package('end;', 4);
  ad_ddl.create_package(l_applsys_schema, 'FND', 'FND_EX_TEST', 'TRUE', 1, 4);
END;
/
```

2. Disable or block application processing that runs Dynamic DDL during online patching.

   This approach can only be used if the dependent functionality can be unavailable for days at a time without disrupting normal customer operations.

   This approach is meant only as a temporary workaround.

   See:

   - Section 2.5.1: Example of Disabling Functionality while Online Patching
   - Section 2.5.2: Blocking a Concurrent Program while Online Patching

Online Patching Compliance Standards

Do not install PL/SQL in on-editioned schemas such as CTXSYS, SYSTEM, SYS, or PUBLIC.

Section 2.2.4: User-Defined Type (Editioned)

For more information on types, see: *Oracle Database PL/SQL Language Reference.*

Definition Standards

Same as that for PL/SQL packages.

Usage Standards

Editioned User-Defined Types may not be used as Column Types or Advanced Queue (AQ) Payload Types. **[minimal]**

To create a User-Defined Type for use as a column type, refer to Section 2.4.2: User-Defined Type (UDT) (Non-Editioned) later in this document.

Dynamic DDL Standards

Same as PL/SQL packages.

Due to a database limitation, an existing Editioned User-Defined Type may not be evolved. This means that you cannot alter the structure of the type using the "ALTER TYPE ..." statement. Use "CREATE OR REPLACE TYPE ..." instead. **[minimal]**

Online Patching Compliance Standards

Due to a database limitation, an existing Editioned User-Defined Type may not be evolved. This means that you cannot alter the structure of the type using the "ALTER TYPE ..." statement. Use "CREATE OR REPLACE TYPE ..." instead. **[minimal]**

Section 2.2.5: PL/SQL Trigger

For information on PL/SQL triggers, see: *Oracle Database PL/SQL Language Reference.*

Definition Standards

- The trigger name must be a non-quoted identifier. (dbcc: SECTION-10)

  Note: Non-quoted identifiers may only use the following characters: A-Z 0-9 _ # $. Quoted identifiers are reserved for use by the technology components.

- A table trigger must be on the editioning view (EV), not the table, if the editioning view exists. (dbcc: SECTION-13) **[minimal]**

  Triggers on tables will be automatically moved to the corresponding editioning view during the Release 12.2 upgrade. For information on editioning views, see: *Oracle Database Advanced Application Developer's Guide*.

  Tip: The simplest way to comply with this standard is to create triggers "on" the APPS table synonym. The trigger will be created on whatever the synonym points to, which will be the editioning view if it exists or the table if there is no editioning view.

  Note: EV triggers reference logical columns in the editioning view. The editioning view maps logical columns to the correct physical table columns in each edition.

Dynamic DDL Standards

If the runtime application creates, replaces, or drops a trigger while a patch edition exists, then the same action must be run in the patch edition. **[minimal]**

This requirement can be satisfied in either of two ways:

1. Replicate the run edition dynamic DDL in the patch edition.

   Use AD_DDL, which will replicate run edition DDL for triggers in the patch edition automatically.

   For example:

```
/* Code example:  Create a trigger using AD_DDL */

declare
  L_APP   varchar2(8);
  L_NAME  varchar2(30);
  L_STMT  varchar2(32000);
begin
  l_app  := 'FND';
  l_name := 'FND_EXAMPLE_TRG';
  l_stmt :=
    'create or replace trigger '||l_name||' '||
    'before insert or update on fnd_user for each row '||
    'begin '||
    '  ad_zd_log.message(''test'', ''STATEMENT'', ''hello!''); '||
    'end;';

  ad_ddl.do_ddl(
    ad_zd.applsys_schema,  -- applsys schema name
    l_app,                 -- application short name for your product
    ad_ddl.create_trigger, -- statement type
    l_stmt,                -- statement text
    l_name                 -- trigger name
  );
end;
/
```

2. Disable or block application processing that runs dynamic DDL during online patching. This approach can only be used if the dependent functionality can be unavailable for days at a time without disrupting normal customer operations.

   See:

   - Section 2.5.1: Example of Disabling Functionality while Online Patching
   - Section 2.5.2 Blocking a Concurrent Program while Online Patching

Online Patching Compliance Standards

Do not install triggers in non-editioned schemas such as CTXSYS, SYSTEM, SYS, or PUBLIC.

Section 2.2.6: Synonym

For information on managing synonyms, see: *Oracle Database Administrator's Guide*.

Definition Standards

- Most APPS synonyms are automatically created by the adop patching tool. Do not create, modify, or drop automatically managed synonyms unless directed by this standard.
- A table synonym must point to the editioning view, not the table, if the editioning view exists. (dbcc: SECTION-14) **[minimal]**
- A synonym must point to an object. (dbcc: SECTION-15) **[minimal]**
- Do not create public synonyms.

Usage Standards

Use APPS synonyms to reference objects in EBS product schemas. Do not reference objects in Oracle E-Business Suite product schemas directly. **[minimal]**

Note: The APPS synonym layer provides a Logical Schema view over the physical objects in the Oracle E-Business Suite product schemas. The logical schema is considered the stable public interface, while the physical implementation may change without warning.

- The Logical name for an object is its APPS synonym name.
- The APPS synonym points to the Oracle E-Business Suite product schema object that physically implements the logical name.
- The APPS synonym is editioned and may point to different objects in different editions of the application.
- The APPS synonym for a table points to the editioning view instead of the table, if the editioning view exists.

Dynamic DDL Standards

If the running application creates, replaces, or drops a synonym while a patch edition exists, then the same action must be run in the patch edition. **[minimal]**

This requirement can be satisfied in either of two ways:

1. Replicate the run edition dynamic DDL in the patch edition.

   Use AD_DDL, which will replicate run edition DDL for synonyms in the patch edition automatically.

   Example:

```
/* Code example:  Create a synonym using AD_DDL */

declare
  L_APP   varchar2(8);
  L_NAME  varchar2(30);
  L_STMT  varchar2(32000);
begin
  l_app  := 'FND';
  l_name := 'FND_USERS';
  l_stmt :=
    'create or replace synonym '||l_name||' for fnd_user';

  ad_ddl.do_ddl(
    ad_zd.applsys_schema,  -- applsys schema name
    l_app,                 -- application short name for your product
    ad_ddl.create_synonym, -- statement type
    l_stmt,                -- statement text
    l_name                 -- view name
  );
end;
/
```

2. Disable or block application processing that runs dynamic DDL during online patching.

   This approach can only be used if the dependent functionality can be unavailable for days at a time without disrupting normal operations.

   See:

   - Section 2.5.1 Example of Disabling Functionality while Online Patching
   - Section 2.5.2 Blocking a Concurrent Program while Online Patching

Online Patching Compliance Standards

Do not install synonyms in non-editioned schemas such as CTXSYS, SYSTEM, SYS, or PUBLIC.

Section 2.2.7: Virtual Private Database (VPD) Policy

For information on Oracle Virtual Private Database, see: *Oracle Database Security Guide*.

Definition Standards

A VPD Policy must be on the editioning view or table synonym, not the table. (GSCC File.Sql.80, dbcc: SECTION-16) **[minimal]**

VPD Policies on tables will be automatically moved to the corresponding editioning view during the Release 12.2 upgrade.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

Tip: You can add or drop VPD policies by calling the FND_ACCESS_CONTROL_UTIL package from a SQL script (&phase=pdt or &phase=en).

Section 2.3: Effectively-Editioned Objects

These object types are non-editioned at the database level, meaning they have a single definition that is visible to all database editions. But for each object in this category, the Online Patching tools implement special support so that the object can be patched without impacting the running application. New restrictions and standards apply to each effectively-editioned object type.

Section 2.3.1: Table (Ordinary)

An ordinary table is created, altered or dropped during application patching. In contrast, a dynamic table is created, altered, or dropped by the application at runtime. The standards in this section apply to ordinary tables only.

In order to implement effectively-editioned support for ordinary tables, the online patching technology installs and maintains a new editioning view layer over each table. The editioning view maps logical column names used by the application to the actual storage columns used to hold those attributes in each edition. Although the editioning view is maintained automatically during online patching, developers who create or alter tables by hand in a development database must follow new procedures to maintain the editioning view. Please read the complete description of effectively-editioned table development and patching procedures in Section 1.4.3.1 Tables above.

For more information on guidelines for managing tables, see: *Oracle Database Administrator's Guide*.

Definition Standards

- A table name must not end with the '#' character. (GSCC File.Gen.34, dbcc: SECTION-17)

  For tables in schemas that will be editions enabled:

  Tables in the same schema must have table names that are unique within the first 29 characters. This means that a schema cannot contain two tables with table names that only differ in the 30th character of the table name.

  If you plan to use online patching for custom tables, then column names in a custom table must be unique within the first 28 characters. If you have column names that differ only in the 29th or 30th character, then you will not be able to patch these columns while keeping the same logical column name using online patching. Such columns can only be patched in downtime or by choosing a different (shorter) column name and adjusting application code to reference the new column name.
- A table name must be unique within the first 29 bytes. (GSCC File.Xdf.4, dbcc: SECTION-18)
- A table must be owned by an Oracle E-Business Suite product schema or custom product schema, not APPS. (GSCC File.Gen.35, dbcc: SECTION-19)
- A table must have an editioning view. (dbcc: SECTION-20)

  Note that tables created via patching tools such as XDF/ODF will get an editioning view automatically.

  Also note that tables created via direct DDL get the editioning view by calling AD_ZD_TABLE.UPGRADE. Review Section 1.4.3.1 Tables for more information.
- An APPS synonym (serving as the logical table name) must point to the editioning view.

  Note: The synonym is automatically installed by the AD_ZD_TABLE.UPGRADE procedure.
- A table must not be "index-organized" (*create table ... ORGANIZATION INDEX*). Index-organized tables cannot be patched using online patching.
- A column name must be a nonquoted identifier. For information on nonquoted identifiers, see: *Oracle Database SQL Language Reference*.
- A base column name may only use '#' as the last character. (GSCC File.Gen.36, dbcc:SECTION-21)

  Note: The base column name is the original column name before any revised columns are created. The base column name is used as the logical column name exposed through the editioning view.
- A base column name should be 28 bytes or fewer. (GSCC File.Xdf.4, dbcc: SECTION-22)

  Note: Online patching currently does not support patching columns that violate this standard. If a violating column must be patched, then it must be replaced with a compliant column (that is, with a shorter name) as part of the patch. Application code will need to be updated to reference the new shorter column name.
- The column type must be a built-in type or a user-defined type owned by a non-editioned user. (GSCC File.Gen.37, dbcc: SECTION-23) **[minimal]**
- The column type must not be LONG or LONG RAW. See: LONG to CLOB Conversion Procedures, *Oracle E-Business Suite Developer's Guide*. (GSCC File.Xdf.4, dbcc: SECTION-24)

  Note: When directly altering a column from LONG to CLOB in a development database, you must manually rebuild indexes on that table by hand. But do NOT include manual index rebuild scripts in a patch, index rebuild will be done automatically by XDF/ODF when needed.
- The column type should not be ROWID. (GSCC File.Xdf.4, dbcc: SECTION-25) **[minimal]**

  ROWID references can become invalid if the target table is patched, loaded, or rebuilt. It is not safe to store ROWID references across an online patching cutover boundary.
- If adding a column with a default value, the column must be not null. **[minimal]**

  Explanation: Oracle Database Advanced Compression requires that new columns with a default value be not null.

Usage Standards

- Query/DML statements must access tables via the table synonym or editioning view. (GSCC File.Gen.41, dbcc: SECTION-26) **[minimal]**

  Tip: Avoid using table aliases that are also schema names; this practice may generate false positives on GSCC checks.
- If you query, display, or store table column names in your application runtime code, you should use logical column names rather than physical column names in most cases. **[minimal]**
  - Follow the guidelines in Section 2.3.7 Logical versus Physical Table Column Guidelines for runtime application code.

    Warning: Some dictionary views (such as ALL_TAB_COLUMNS) contain information for both Logical and Physical table columns, depending on whether you query the editioning view or base table data. Consult Section 2.3.7 Logical versus Physical Table Column Guidelines for details.
  - Review Oracle E-Business Suite objects that reference ALL_TAB_COLUMNS or DBA_TAB_COLUMNS and ensure that your query is selecting the correct logical or physical column information for your purposes.
- DDL statements such as TRUNCATE will not work on an APPS table synonym that points to an editioning view. To truncate a table, you must supply the actual base table (owner.table_name) in the truncate command. **[minimal]**

  Tip: Generate the truncate command dynamically based on the APPS table synonym name as follows:

  ```
  select 'truncate table '||table_owner||'.'||rtrim(table_name, '#') truncate_command from user_synonyms where synonym_name=input_synonym_name;
  ```
- Do not rely on the ordering of columns within the table, and do not assume that additional columns are not present. **[minimal]**
  - Specify an explicit column list when selecting
    - Bad: `select * from table`
    - Good: `select col1, col2, ... from table`
  - Specify an explicit column list when inserting
    - Bad: `insert into table values (val1, val2, ...)`
    - Good: `insert into table (col1, col2, ...) values (val1, val2, ...)`

Dynamic DDL Standards

- Application-managed tables are tables that are created and maintained by application logic during normal application runtime:
  - Application-managed (dynamic) tables must not have an editioning view.
  - Do not modify application-managed tables in an online patch. (GSCC File.Gen.38)
- Ordinary tables are created and maintained by online patching (and will have an editioning view).
  - If the application logic modifies an ordinary table at runtime, it must use the AD_DDL interface to run the dynamic DDL. **[minimal]**
  - Do not modify ordinary tables in the run edition while a patch edition exists.**[minimal]**
  - Do not generate dynamic staging tables for the purpose of partition exchange with a partitioned table. A dynamically-created staging table structure may fall out of synchronization with the partitioned table during online patching. Partitioned Tables may only exchange partitions with a permanent staging table that is maintained with the same structure as the partitioned table during online patching. See Section 1.4.4.2 Partition Exchange for detailed procedures.

Online Patching Compliance Standards

- Do not run DML statements (insert/update/delete/truncate) against existing tables from the patch edition.
  - Exception: You can run DML against prepared Seed Data Tables. See Seed Data Tables for details.
  - You can run safe "Data Fixer" DML against existing tables by switching to the Run Edition in your SQL script. Refer to Section 1.4.4.1 Data Fixer Patch for details.
- Patch the table definition using ODF or XDF. (GSCC File.Sql.81)
  - Note: XDF/ODF will automatically install and maintain the editioning view for an ordinary table. The editioning view will map each logical column name to the latest revision of that column.
  - Example: Table "FND_PROFILE_OPTIONS" has an editioning view called "FND_PROFILE_OPTIONS#".
- Do not alter or drop an existing column definition, or update existing column data in a way that is incompatible with the running application.
- To patch the data type or content of an existing logical column, create a new revised column and populate it with a forward crossedition trigger. See Section 1.4.3.1 Tables for detailed examples.
  - A revised column name has the form: `<logical_column_name>#<version_tag>`
    - <Version_tag> is a string of the form: [0-9A-Z]+.
    - Version tags are compared using the SQL '>' operator. A "greater" tag is a later version.
      - Example: Column "AMOUNT" is replaced by revised column "AMOUNT#1".
      - Example: Column "AMOUNT#1" is replaced by revised column "AMOUNT#2".
  - Populate a revised column using a forward crossedition trigger (FCET).
    - The FCET must be owned by the APPS user.
    - The FCET name should be of the form <table_name>_F<change_number>. Example: "FND_PROFILE_OPTIONS_F3"
    - The FCET must be created with the "disable" option.
    - The FCET for the second and subsequent change to a given table must specify that it "follows <previous_fcet_name> " in the trigger definition. For information on the FOLLOW clause, see: *Oracle Database PL/SQL Language Reference*.
      - Note: if you create an FCET with the "follows <previous_fcet_name>" clause, and the previous FCET does not exist, the trigger will get compilation errors. You can ignore these, as the online patching tool will automatically create a dummy previous FCET when the trigger is applied.
      - For ease of development, you can leave out the "follows ..." clause during initial creation, verify that the trigger compiles without error, and then add the "follows ..." clause when you are otherwise ready to test.
    - The FCET creation script must use standard dbdrv comments as follows:
      - `REM dbdrv: sql ~PROD ~PATH ~FILE none none none sqlplus &phase=ccet checkfile:~PROD:~PATH:~FILE &un_<prod>`
      - `REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql none none none sqlplus &phase=acet checkfile:~PROD:~PATH:~FILE:fcet <FCET_NAME>`
    - Tip: Use the Forward Crossedition Trigger template:

```
REM ---- Create FCET ----
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=ccet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_<table_owner_app_short_name>
REM ---- Apply FCET ----
REM dbdrv: sql ad patch/115/sql AD_ZD_TABLE_APPLY.sql \
REM dbdrv:   none none none sqlplus &phase=acet \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE:fcet <table_name>_F<change_number>

REM Copyright (c) 2015 Oracle Corporation, All Rights Reserved
REM $Header$
REM <table_name>_X<change_number>.sql
REM <description of change>

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace trigger <table_name>_F<change_number>
  before insert or update on &1..<table_name>
  for each row  forward crossedition
  /* follows <previous_fcet> */ disable
begin
  <upgrade logic>
end;
/

commit;
exit;
```

  - Change any index definitions that referenced the original column to instead reference the revised column.

    Note: creation of revised unique indexes must be deferred until after the revised column is populated, &phase=dfr.
  - When patching a new or revised column with not null constraint to an existing table, specify a default value for the column.

    Tip: Use a default value which clearly indicates that the column is not populated with real data yet. Examples:
    - varchar2: '*NULL*'
    - number: -1
    - date: to_date('01/01/1900', 'DD/MM/YYYY')
  - Do not drop an existing table column directly. Drop a logical column using AD_ZD_TABLE.OBSOLETE_COLUMN.
    - Create a SQL script that runs the following statement:

      `exec ad_zd_table.obsolete_column('TABLE_OWNER', 'TABLE_NAME', 'COLUMN_NAME')`

      Note: Do not hard-code table owner; the table schema name should be a parameter to your script.

      Note: The script should run in a phase immediately after the normal phase in which your XDF/ODF for the same table runs.
    - The logical column will be removed from the EV, but the table column will remain until the table is re-built without unused columns.
  - Do not rename an existing table. This is currently not supported.
  - Do not drop an existing table until the cleanup phase of the patching process.
    - Drop the editioning view during the APPLY phase, and store a deferred DDL to drop the table during the cleanup phase. [dbcc: SECTION-12]
    - For an example of a deferred drop, see: Section 2.5.4: Example of a Deferred Drop.
  - When patching a Partitioned Table that has a partition exchange staging table, also patch the staging table in the same way.

Section 2.3.2: Seed Data Table

Seed Data Tables contain data owned by the application, and so the contents of these tables may be changed during online patching. Seed data tables must implement the Editioned Data Storage architecture, which allows the table to store separate run and patch edition copies of seed data, so that changes from online patching do not affect application runtime. Please read the full description for Seed Data Table development and patching procedures above in Seed Data Tables.

Definition Standards

- Ordinary Table Definition Standards also apply to Seed Data Tables.
- A Seed Data Table must implement Editioned Data Storage.
  - Seed Data Tables that exist as of the Oracle E-Business Suite Release 12.2.0 upgrade must be upgraded to Editioned Data Storage during the Oracle E-Business Suite 12.2.0 Upgrade.
    - Each product must create a seed data table upgrade script using the template below. Add one call to AD_ZD_SEED.UPGRADE for each seed data table owned by the product. Do not call the UPGRADE procedure for tables outside of your product; products must only upgrade the tables that they actually own.
    - Template Seed Data Table Upgrade Script (for existing tables during the Oracle E-Business Suite 12.2.0 upgrade only):

```
REM $Header: $
REM dbdrv: sql ~PROD ~PATH ~FILE none none none sqlplus &phase=last+95 \
REM dbdrv: checkfile:~PROD:~PATH:~FILE

REM /*+======================================================================+
REM  |             Copyright (c) 2012 Oracle, California, USA               |
REM  |                         All rights reserved.                         |
REM  +======================================================================+
REM  | Seed Data Table Upgrade for Online Patching:
REM  | Converts Seed Data Tables to support Editioned Data Storage
REM  +======================================================================*/
```

```
SET VERIFY OFF
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR  EXIT FAILURE ROLLBACK;

exec AD_ZD_SEED.UPGRADE('<seed_data_table_name>')
...

commit;
exit;
```

- Example Seed Data Table Upgrade Script:

```
REM $Header: $
REM dbdrv: sql ~PROD ~PATH ~FILE none none none sqlplus &phase=last+95 \
REM dbdrv: checkfile:~PROD:~PATH:~FILE

REM /*+======================================================================+
REM  |            Copyright (c) 2012 Oracle, California, USA                |
REM  |                         All rights reserved.                         |
REM  +======================================================================+
REM  | Seed Data Table Upgrade for Online Patching:
REM  | Converts Seed Data Tables to support Editioned Data Storage
REM  +======================================================================*/

SET VERIFY OFF
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR  EXIT FAILURE ROLLBACK;

exec AD_ZD_SEED.UPGRADE('FND_NEW_MESSAGES')
exec AD_ZD_SEED.UPGRADE('FND_APPLICATION')
exec AD_ZD_SEED.UPGRADE('FND_APPLICATION_TL')

commit;
exit;
```

- Note: this requirement applies to the upgrade of existing seed data tables during the Oracle E-Business Suite 12.2.0 Upgrade only. An existing Ordinary Table may not be upgraded to be Seed Data Table in an online patch. It is possible to deliver a new Seed Data Table in an online patch. Refer to the Patching Standards below.

  - When creating a new Seed Data Table in a development database, first create the ordinary table and then call the AD_ZD_SEED.UPGRADE procedure manually. Refer to the section Upgrade table to support Editioned Data Storage for more information.

- A seed data table must have a unique index. (dbcc: SECTION-32)
  - Without a unique index, run edition updates to seed data cannot be synchronized to the patch edition copy and will be lost.
  - This standard does not apply if the table is read-only to the runtime application.

Usage Standards

- Ordinary table usage standards also apply to seed data tables.
- "INSERT ALL" (as documented in *Oracle Database SQL Language Reference*) is no longer supported against seed data tables. Re-code this statement as separate insert statements.**[minimal]**
- Avoid use of the ZD_EDITION_NAME column in application logic **[minimal]**:
  - Do not select, insert or update the ZD_EDITION_NAME column, or reference the column value in any way.
  - Do not include ZD_EDITION_NAME in views, APIs, or any application logic.
  - Do not include ZD_EDITION_NAME in user interfaces, reports, or any other user display.
  - Do not include ZD_EDITION_NAME in BC4J entity objects, view objects, or any other automatically generated artifact. You might need to explicitly remove the ZD_EDITION_NAME column from generated code.
- Do not access seed data rows by ROWID (`where rowid = <local_variable>`) from any code that can run in the patch edition.
  - Code that can run in the patch edition includes crossedition trigger logic, seed data loader logic, SQL scripts in patches, and any code run by the patching tool itself.
  - In the patch edition, any locally held seed data ROWID may become invalid at any time, if the related seed data table is prepared in a different session.
- Do not under any circumstances disable the ZD_SEED VPD policy or generated maintenance trigger on a seed data table.

Dynamic DDL Standards

Not applicable.

Section 2.3.3: Seed Data Loader

Definition Standards

- FNDLOAD Configuration Files (LCT files) must include a PREPARE statement for each entity listing the seed data tables that the UPLOAD command loads into.

  Syntax:

```
PREPARE <entity>
TABLE <seed_data_table_name1>
TABLE <seed_data_table_name2>
...
```

  <*seed_data_table_name*> is the name of the APPS synonym for the seed data table.

  The ordering of UPLOAD / DOWNLOAD / PREPARE statements in an LCT file is not important.

  Note that an entity does not require a PREPARE statement if any of the following is true:

  - The entity is not stored in a seed data table.
  - The entity is not patched during online patching or does not support UPLOAD.
  - The seed data tables loaded by a child entity are prepared by the parent entity.
  - The PREPARE call is already handled by the upload logic.

  You should only list seed data tables that will be loaded. For example, if the upload logic initially stores data in temporary tables, the temporary tables should not be listed in the PREPARE statement. Only seed data tables can be prepared.

  Example:

```
# $Header: wfmlrp.lct 120.0 2005/05/07 16:19:43 appldev ship $
COMMENT = "dbdrv: exec fnd bin FNDLOAD bin &phase=daa+52 checkfile:~PROD:~PATH:~FILE &ui_apps 0 Y UPLOAD @FND:patch/115/import/wfmlrp.lct @~PROD:~PATH/~FILE"

DEFINE MAILERPARAMS
   KEY  NAME       VARCHAR2(12)
   KEY  PARAMETER   VARCHAR2(30)
   BASE VALUE       VARCHAR2(200)
   BASE REQUIRED    VARCHAR2(1)
   BASE CALLBACK    VARCHAR2(60)
   BASE ALLOWRELOAD VARCHAR2(1)
END MAILERPARAMS

DOWNLOAD MAILERPARAMS
   "select NAME, PARAMETER, VALUE, REQUIRED, CB, ALLOW_RELOAD
    from WF_MAILER_PARAMETERS
    where NAME = :NAME"

###
### Do NOT call AD_ZD_SEED.PREPARE from the UPLOAD statement of an LCT file.
### Use the LCT PREPARE statement instead (see below)
###
UPLOAD MAILERPARAMS
   "begin
      wf_mailer_parameter.PutParameter(:NAME, :PARAMETER,:VALUE, :REQUIRED, :CALLBACK, :ALLOWRELOAD);
    end;"

###
### New for Online Patching
###
PREPARE MAILERPARAMS
  TABLE WF_MAILER_PARAMETERS
```

- Other Seed Data Loaders and scripts (not implemented using FNDLOAD) must manually call the AD_ZD_SEED.PREPARE procedure before loading data into a seed data table. Note: this requirement does NOT apply to scripts and loaders that will only be used for the Oracle E-Business Suite Release 12.2 upgrade.

  Syntax: AD_ZD_SEED.PREPARE('<*table_name*>');

  Note: `<seed_data_table_name>` is the name of the APPS synonym for the seed data table.

  The AD_ZD_SEED.PREPARE procedure cannot be called from scripts using the "dbdrv: sql ... **sql**" execute method. Use "dbdrv sql ... **sqlplus**" instead.

  - Bad Example: `dbdrv: sql ~PROD ~PATH ~FILE none none none` **`sql`** `&phase=upg+14 ...`
  - Good Example: `dbdrv: sql ~PROD ~PATH ~FILE none none none` **`sqlplus`** `&phase=upg+14 ...`
  - Good Example: `dbdrv: sql ~PROD ~PATH ~FILE none none none` **`sqlplus_single`** `&phase=upg+14 ...`

Example SQL script with AD_ZD_SEED.PREPARE procedure call:

```
REM $Header: $
REM dbdrv: sql ~PROD ~PATH ~FILE none none none sqlplus_single &phase=daa \
REM dbdrv: checkfile(115.2=120.1):~PROD:~PATH:~FILE &un_fnd &pw_fnd

REM /*+======================================================================+
REM |           Copyright (c) 2012 Oracle, Belmont, California, USA         |
REM |                       All rights reserved.                           |
REM +======================================================================+
REM |
REM | NAME
REM |   ldrexample.sql
REM |
REM | DESCRIPTION
REM |  This script adds a policy to the wf_signature_policies
REM +======================================================================*/

SET VERIFY OFF;
WHENEVER SQLERROR EXIT FAILURE ROLLBACK
WHENEVER OSERROR EXIT FAILURE ROLLBACK

--
-- NEW FOR ONLINE PATCHING
--
exec ad_zd_seed.prepare('WF_SIGNATURE_POLICIES')

insert into WF_SIGNATURE_POLICIES
    (SIG_POLICY, SIG_REQUIRED, FWK_SIG_FLAVOR,
     EMAIL_SIG_FLAVOR, RENDER_HINT, DEFAULT_POLICY)
  select 'DEFAULT', 'N', Null, Null, Null, 'Y'
    from dual
   where not exists
     (select 1
        from WF_SIGNATURE_POLICIES
       where SIG_POLICY = 'DEFAULT');

commit;
exit;
```

- Seed data upload logic run during online patching must not reference materialized views, since materialized views are non-editioned objects and will be defined according to the run edition rather than the patch edition.
- Seed data upload logic must not reference seed data rows by ROWID. See Section 2.3.2 Seed Data Table above for details.

Usage Standards

No special considerations.

Dynamic DDL Standards

Not applicable.

Section 2.3.4: Index

For information on indexes, see: *Oracle Database Concepts*.

Definition Standards

- The index name must contain an underscore ('_'). (GSCC File.Gen.39, dbcc: SECTION-33)
- The unique index on a seed data table must include ZD_EDITION_NAME. (dbcc: SECTION-35)

  Note: This rule will be implemented automatically when you call "ad_zd_seed.upgrade" on your seed data table, but if you add a new unique index to an existing seed data table you will need to include the ZD_EDITION_NAME column in your index definition.
- The index key length should be less than 3125 bytes. (dbcc: SECTION-37)

  The index key length is the sum of the column lengths for each column in the index, plus one byte for each column.

  If the index key length is greater than 3125 bytes, then the index cannot be revised using online index definition, and a full table lock will be held during index creation.
- A function-based index must not reference editioned Oracle E-Business Suite objects (built-in database functions such as "UPPER()" are acceptable). **[minimal]**

Dynamic DDL Standards

The "CREATE INDEX ... ON ..." statement must specify the fully qualified table name, not the APPS table synonym.

- Good Example: `create index SOME_TABLE_N1 on SCHEMA.SOME_TABLE ...`
- Bad Example: `create index SOME_TABLE_N1 on SOME_TABLE ...`

Online Patching Compliance Standards

- Deliver the index definition using ODF or XDF.

  Note: During apply, new or revised indexes will be initially created with an alternate name. During cutover, the obsolete indexes will be dropped and new or revised indexes will be renamed to the specified names.
- Do not drop an existing unwanted index until the Cutover phase of patch execution For an example of a deferred drop, see: Section 2.5.4: Example of a Deferred Drop.
- It is not possible to create a new or revised index with the same column list as an existing index on the same table. Attempting to do this results in ORA-01408: such column list already indexed.
  - This issue may come up if attempting to change an existing index from unique to non-unique or vice-versa.
  - The only way to change the definition of an index while keeping the same column list is to execute the change as a cutover DDL.

Section 2.3.5: Integrity Constraint

For more information on integrity constraints, see: *Oracle Database Concepts*.

Definition Standards

- The constraint name must contain an underscore ('_').(dbcc: SECTION-38)
- Do not create a primary key constraint on a table. Create a unique index or unique constraint instead.

  An existing primary key constraint will work, but it is not possible to create a revised primary key constraint in an online patch.
- Do not create a unique key constraint on a table. Create a unique index instead.

  An existing unique key constraint will work, but if the key is ever revised the unique key constraint will be dropped along with any foreign key constraints that reference it. Unique constraints are not automatically revised by online patching.
- Do not create a foreign key constraint on a table. If the referenced primary or unique key is dropped, then the foreign key will also be dropped. Foreign key constraints are not automatically revised by online patching.
- A foreign key constraint cannot be created to a seed data table. (dbcc: SECTION-39) **[minimal]**

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

- Do not attempt to patch an existing primary key constraint. Create a unique index instead.

  Note that Online Patching will automatically drop constraints on obsolete columns during cutover, so you do not need to drop the old primary key yourself.
- Deliver the constraint definition using ODF or XDF. (GSCC File.Sql.81)

  New or revised constraints will be initially created as disabled, and will be enabled during cutover. For more information on using XDF (XML Definition File) features, see the *Oracle E-Business Suite Developer's Guide*.

Section 2.3.6: Materialized View (MV)

A materialized view is a non-editioned object type, and therefore a materialized view cannot directly reference editioned objects. To avoid this limitation, Oracle E-Business Suite online patching technology implements a new effectively-editioned materialized view compound object. Application developers create and maintain the materialized view definition (query) in an ordinary view. The online patching technology then automatically maintains a corresponding materialized view implementation that is legal for editioned databases. Please read the full description of effectively-editioned materialized view development and patching procedures in Section 1.4.3.4: Materialized Views.

Note: Oracle Database 12.1.0.2 introduces a new feature to materialized view create syntax called the the Evaluation Edition clause. This feature allows a native materialized view definition to reference editioned objects in the specified evaluation edition. Customers who wish to create custom materialized views may use native materialized views with the evaluation edition clause instead of following the "effectively-editioned materialized view" standards in this section. For more information on the Evaluation Edition clause, refer to *Database SQL Language Reference* for your database version.

Definition Standards

Definition Standards:

- A materialized view name must be unique within the first 29 bytes. (dbcc: SECTION-40) **[minimal]**
- A materialized view definition must be stored in an ordinary view called `MV_NAME||'#'`. (dbcc: SECTION-41, dbcc: SECTION-42) **[minimal]**
  - Create or replace the materialized view definition as an ordinary view called `mv_name||'#'`.
  - Test the materialized view definition for accuracy before generating the materialized view implementation.
  - For example:

```
create or replace view FND_EXAMPLE_MV# as select ... ;
select * from fnd_example_mv#;
```

- The materialized view implementation is automatically generated from the materialized view definition using the AD_ZD_MVIEW.UPGRADE procedure. **[minimal]**

  Syntax is `exec ad_zd_mview.upgrade(<MV_OWNER>, <MV_NAME>)`

  Do not attempt to directly create or replace the materialized view implementation. To recreate an MV implementation, call the AD_ZD_MVIEW.UPGRADE procedure.
- A materialized view definition must specify a column alias for each item in the select list. **[minimal]**
  - Failure to specify a column alias may cause the error ORA-00998 "must name this expression with a column alias".
  - Example: change `select sum(EMP.SALARY), ...` to `select sum(EMP.SALARY) SUM_EMP_SALARY, ...`
- A materialized view definition must not reference editioned PL/SQL functions. **[minimal]**
  - If the materialized view definition references an editioned PL/SQL function, then the materialized view implementation will fail to generate and the materialized view will be unusable.
  - For examples of replacing PL/SQL function calls with equivalent SQL in materialized views, see: Section 2.5.3: Examples of SQL Replacements for PL/SQL Functions.
- A materialized view should use 'REFRESH FORCE' instead of 'REFRESH FAST'. The 'FORCE' option allows the materialized view to fall back to using a complete refresh in situations where the fast refresh is not possible.

  See: *Oracle Database SQL Language Reference* for more information on the "REFRESH FORCE" option.
- If the materialized view implementation content must be automatically refreshed after patching, then you must include the '/*AUTOREFRESH*/' comment tag in the materialized view definition query.
  - Do not specify the /*AUTOREFRESH*/ tag for large materialized views that will take a long time to refresh. For these cases use a concurrent program to refresh the materialized view after patching cutover.
  - Example: `create or replace view FND_EXAMPLE_MV# as select /*AUTOREFRESH*/ ... ;`

Usage Standards

Do not assume that fast refresh is always possible. After an online patch, complete refresh may be required. When refreshing a materialized view, us the 'FORCE' clause instead of 'FAST'.

See: *Oracle Database SQL Language Reference* for more information on the 'FORCE' option.

Dynamic DDL Standards

Use AD_MV to run dynamic DDL for materialized views. **[minimal]**

Here is an example of creating a materialized view using the AD_MV package:

```
--
-- Code Example:  Create a materialized view using AD_MV interface.
--
-- Note:
--    when run in the Run Edition, the MV is created immediately.
--    when run in the Patch Edition, the MV is generated at CUTOVER.
--


begin

  -- Create MV
  ad_mv.create_mv('FND_EXAMPLE_MV',
    'create materialized view FND_EXAMPLE_MV '||
    '  tablespace '||ad_mv.g_mv_data_tablespace||' '||
    '  build deferred refresh on demand as '||
    'select  '||
    '    upper(oracle_username) USERNAME '||
    '  , decode(read_only_flag,''C'',''pub'',''E'',''applsys'',''U'',''apps'') USERTYPE '||
    'from  fnd_oracle_userid '||
    'where read_only_flag in (''C'',''E'',''U'') ');

 end;
```

Online Patching Compliance Standards

Deliver materialized view using XDF. ADOP and XDF work together to implement effectively editioned materialized views. (GSCC File.Sql.82) **[minimal]**

For more information on using XDF (XML Definition File) features, refer to the *Oracle E-Business Suite Developer's Guide*.

Do not attempt to upgrade, refresh or access a materialized view implementation in the patch edition. Although the MV implementation is visible to the patch edition, it continues to implement the run edition of the definition until the cutover phase. MV implementations are automatically regenerated as needed at the cutover phase.

- If an online patch must manually refresh the MV implementation contents, submit a concurrent request to do the refresh. The concurrent request will be run after cutover and therefore after the MV implementation has been regenerated.
- If the MV definition specifies the /*AUTOREFRESH*/ comment tag, then the MV contents will be automatically refreshed whenever the MV implementation is regenerated.

Do not drop an obsolete materialized view until the cleanup phase of patch execution.

Section 2.3.7: Logical versus Physical Table Columns

In Oracle E-Business Suite Release 12.2, the online patching architecture introduces a new editioning view (EV) layer over developer-managed tables. The EV layer provides a stable logical view of the table information to the application, even though the names of the physical storage columns will change with online patching. The EV maps each logical column name to the latest revised storage column for that attribute. The runtime application is expected to access table data via this logical view.

Effectively-editioned tables are patched using online patching techniques that will cause the physical storage column names to be different from the logical column names over time. This logical-to-physical column name mapping is transparent to the application for normal query and DML statements, but if the application queries database data dictionary views, then it must take care to select the correct (logical or physical) column names depending on the purpose for getting the names.

- If column names are selected for the purpose of constructing query or DML statements (select, insert, update, or delete) then you must select logical column names.
- If column names are selected for the purpose of constructing table DDL statements (for example, to create an index), then you must select physical column names. Normally, application runtime code should not be modifying tables that are managed through online patching, so this should be a rare situation.

Note that the distinction between logical and physical columns only exists for table columns where the table has an editioning view. If you are getting column information for any object that does not have an editioning view then you do **not** need to make any changes. Queries that do not need to be changed include:

- Queries that run as part of the Oracle E-Business Suite Release 12.2 upgrade. During the main upgrade phase editioning views are not yet installed, and so all existing upgrade scripts will work correctly.
- Queries for view column information.
- Queries for table column information, where the table does not have an editioning view. This category includes:
  - Temporary tables.
  - Application-managed (dynamically generated) tables. Application-managed tables are database tables that are dynamically created and managed by application logic during runtime. Whenever the application generates such tables, it is expected that the application will also modify, replace, or drop the tables as needed during application runtime. It is also expected that application-managed tables will not be directly modified by online patches.
  - Advanced Queue tables.
  - Materialized view container tables.
  - Index-organized tables.
  - Any other exception tables that do not have an editioning view.

Database Data Dictionary Views involving Columns

MIXED USE Dictionary Views

Dictionary views in this category contain information for both logical and physical table columns, depending on how the view is queried. You must examine each reference to these dictionary views and determine whether you intend to select logical or physical column information.

- ALL_TAB_COLUMNS, DBA_TAB_COLUMNS, USER_TAB_COLUMNS
- ALL_TAB_COLS, DBA_TAB_COLS, USER_TAB_COLS

- ALL_COL_COMMENTS, DBA_COL_COMMENTS, USER_COL_COMMENTS
- ALL_OBJ_COLATTRS, DBA_OBJ_COLATTRS, USER_OBJ_COLATTRS
- ALL_TRIGGER_COLS, DBA_TRIGGER_COLS, USER_TRIGGER_COLS
- ALL_UPDATABLE_COLUMNS, DBA_UPDATABLE_COLUMNS, USER_UPDATABLE_COLUMNS

To select physical (table) column names, select from the dictionary view for a specific physical table name (where table_name = x_table_name). You can use `where table_name not like '%#` to search across all tables while excluding logical views.

To select logical (EV) column names, select from the dictionary view for a specific editioning view name, where table_name = ad_zd_table.ev_view(x_table_name). Another way to get the logical view information is to start with the APPS synonym and join the synonym table_name to the dictionary view to get the correct logical view information. The APPS synonym will point to the EV if one exists, or directly the physical table if there is no EV. This will provide correct logical column results in either case.

```
/*
** This query returns logical columns for the table pointed to by APPS
** table synonym X_SYNONYM_NAME.  Compatible with old releases.
*/
select col.column_name
from  user_synonyms syn, all_tab_columns col
where syn.synonym_name = x_synonym_name
  and col.owner      =  syn.table_owner
  and col.table_name = syn.table_name
order by col.column_id
/
```

If you do not know whether you are querying a table or a view, you can union the results from the two possible queries:

```
/*
** This query returns logical columns for X_OBJECT_NAME, and works whether
** the object is an APPS table synonym or view.  Compatible with old releases.
*/
select col.column_name, col.data_type
from  user_synonyms syn, all_tab_columns col
where syn.synonym_name = x_object_name
  and col.owner      = syn.table_owner
  and col.table_name = syn.table_name
union
select col.column_name, col.data_type
from  user_tab_columns col
where col.table_name = x_object_name
/
```

If the application logic must use a physical table name as the input rather than the APPS object name, and you are sure that the table has an editioning view, then you can just convert the table name to the editioning view name with a utility function:

```
*
** This query returns logical columns for effectively editioned table
** X_OWNER.X_TABLE_NAME.  Use this only if you cannot start from
** the APPS synonym name and are sure that the table has an EV.
** This query works on Oracle E-Business Suite Release 12.2 only.
*/
select col.column_name
from  all_tab_columns col
where col.owner      = x_owner
  and col.table_name = ad_zd_table.ev_view(x_table_name)
order by col.column_id
/
```

In the most general (worst) case, your application code may not be able to guarantee that the input table name has an EV. In this case you can outer-join to the EV information to provide logical column names if they exist, or physical columns if they do not:

```
/*
** This query returns the logical columns for table X_OWNER.X_TABLE_NAME.
** If the table has an editioning view, then the editioning view columns are
** returned, otherwise the physical table columns are returned.
** ONLY use this query if you must query by physical table name rather
** than APPS synonym name and the table might not have an EV.
** This query works on Oracle E-Business Suite Release 12.2 only.
*/
select
    col.owner
  , col.table_name
  , decode(ev.view_name, NULL, col.column_name, evc.view_column_name) logical_column_name
  , col.column_name physical_column_name
from
    all_tab_columns col
  , all_editioning_views ev
  , all_editioning_view_cols evc
where col.owner        =   x_owner
  and col.table_name   = x_table_name
  and ev.owner(+)      = col.owner
  and ev.table_name(+) = col.table_name
  and evc.owner(+)     = ev.owner
  and evc.view_name(+) = ev.view_name
  and (ev.view_name is null or evc.table_column_name = col.column_name)
order by col.owner, col.table_name, evc.view_column_id
/
```

Logical (EV) Column Information Only

Dictionary views in this category only provide information about logical columns. You will only need to join to these views if you are trying to map physical column names to logical column names yourself.

- ALL_EDITIONING_VIEW_COLS, DBA_EDITIONING_VIEW_COLS, USER_EDITIONING_VIEW_COLS
- ALL_EDITIONING_VIEW_COLS_AE, DBA_EDITIONING_VIEW_COLS_AE, USER_EDITIONING_VIEW_COLS_AE

Physical (Table) Column Information Only

The following dictionary views provide information about physical table columns. Any use of these views is likely for the purpose of managing the actual physical tables, and it is not expected that such references need to be converted to use logical column names. But it is worth checking.

- ALL_AUDIT_POLICY_COLUMNS, DBA_AUDIT_POLICY_COLUMNS, USER_AUDIT_POLICY_COLUMNS
- ALL_CONS_COLUMNS, DBA_CONS_COLUMNS, USER_CONS_COLUMNS
- ALL_IND_COLUMNS, DBA_IND_COLUMNS, USER_IND_COLUMNS
- ALL_IND_EXPRESSIONS, DBA_IND_EXPRESSIONS, USER_IND_EXPRESSIONS
- ALL_LOBS, DBA_LOBS, USER_LOBS
- ALL_LOB_PARTITIONS, DBA_LOB_PARTITIONS, USER_LOB_PARTITIONS
- ALL_LOB_SUBPARTITIONS, DBA_LOB_SUBPARTITIONS, USER_LOB_SUBPARTITIONS
- ALL_PART_KEY_COLUMNS, DBA_PART_KEY_COLUMNS, USER_PART_KEY_COLUMNS
- ALL_PART_LOBS, DBA_PART_LOBS, USER_PART_LOBS
- ALL_STREAMS_COLUMNS, DBA_STREAMS_COLUMNS, USER_STREAMS_COLUMNS
- ALL_SUBPART_KEY_COLUMNS, DBA_SUBPART_KEY_COLUMNS, USER_SUBPART_KEY_COLUMNS
- ALL_UNUSED_COL_TABS, DBA_UNUSED_COL_TABS, USER_UNUSED_COL_TABS
- ALL_XML_TAB_COLS, DBA_XML_TAB_COLS, USER_XML_TAB_COLS

Section 2.4: Non-Editioned Objects

Non-editioned objects have the same definition shared across application editions. Such objects must be patched carefully to avoid affecting the running application. In general, you must not change the definition of an existing non-editioned object during an online patch. Instead, create a new object with a different name, and change the original APPS synonym to point at the new non-editioned object. The following object types are covered:

- Sequence
- User-Defined Type (Non-editioned)
- Temporary Table
- Advanced Queue
- Application Context
- XML Schema
- Database Link
- External Data File

Section 2.4.1: Sequence

For information on sequences, see: *Oracle Database Administrator's Guide*.

Definition Standards

No special considerations.

Usage Standards

No special considerations.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

Do not alter a sequence in a way that is incompatible with the running application.

Do not drop a sequence until the cleanup phase of the patching process.

For an example of a deferred drop, see: Section 2.5.4: Example of a Deferred Drop.

Section 2.4.2: User-Defined Type (UDT) (Non-Editioned)

For information on types, see: *Oracle Database PL/SQL Language Reference*.

Definition Standards

- The type owner must be the "non-editioned APPS" user: APPS_NE, **[minimal]**
- Create an APPS synonym that points to the APPS_NE type.

Note: Existing User-Defined Types that are referenced by table columns or AQs will be automatically upgraded to conform to Non-Editioned UDT standards during the Release 12.2 upgrade.

Usage Standards

- Non-editioned objects (Columns or Advanced Queue payload definitions) must reference a non-editioned UDT using its fully-qualified name: `APPS_NE.<type_name>` **[minimal]**
- Editioned objects should reference a non-editioned UDT using its APPS synonym.

Dynamic DDL Standards

Not applicable.

Online Patching Compliance Standards

Do not alter an existing non-editioned type, instead create a new type. For example, `<type_name>2`.

Do not drop a non-editioned type until the Cleanup phase of the patching process.

Code Example: Deferred Drop Type

```
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=last \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE SYSTEM

-- Example logic to drop non-editioned types under online patching
--
-- Note:  This script drops example type "SYSTEM.FND_EXAMPLE_TYPE" and
-- dependent type "SYSTEM.FND_EXAMPLE_TAB".  An unwanted type must
-- be dropped during CLEANUP phase.  You must use the FORCE keyword to avoid
-- error ORA-02303: cannot drop or replace a type with type or table dependents.
--
-- Usage
--    @TYPE_DROP_SCRIPT SYSTEM

exec ad_zd.load_ddl('CLEANUP', 'drop type &1..FND_EXAMPLE_TYPE force')
exec ad_zd.load_ddl('CLEANUP', 'drop type &1..FND_EXAMPLE_TAB force')
```

Note: When dropping types with AD_ZD.LOAD_DDL , you must use the "FORCE" keyword in the "DROP TYPE" statement. This is because deferred DDL statements are run in parallel and the ordering is not guaranteed. If you do not use the force keyword you risk getting error "ORA-02303: cannot drop or replace a type with type or table dependents".

Example:

- `ad_zd.load_ddl('CLEANUP', 'drop type &&1..GMF_STEP_TAB force' );`
- `ad_zd.load_ddl('CLEANUP', 'drop type &&1..GMF_STEP_TYPE force' );`

Section 2.4.3: Temporary Table

Definition Standards

A Temporary Table must be owned by an Oracle E-Business Suite product schema, not APPS. (GSCC File.Gen.35, dbcc: SECTION-19)

An APPS synonym (serving as the logical table name) must point to the temporary table.

Note: Unlike ordinary tables, temporary tables do not have editioning views.

Usage Standards

No special considerations.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

- Do not alter or drop an existing temporary table definition during an online patch.

  Attempting to do so will result in the error "ORA-14450: attempt to access a transactional temporary table already in use".
- Patch a temporary table definition by creating a new temporary table with a different name, and pointing the original APPS synonym at the new temporary table.

  This is best accomplished using XDF and the procedure described in Section 1.4.3.3.2: Revise an Existing Temporary Table.
  - Name the revised temporary table <base_table_name><revision tag>

    Example: FND_TEMP_TABLE, FND_TEMP_TABLE1, FND_TEMP_TABLE2, ...
  - Replace the original APPS synonym with one that points to the revised temporary table.
  - If the original temporary table had indexes, create these same indexes (with identical names) on the revised temporary table as well.
  - Extract the revised temporary table definition using xdfgen.pl on the APPS synonym for the temporary table.

    Example: `xdfgen.pl <apps_user> FND_TEMP_TABLE`
  - When a revised temporary table is delivered with XDF, the revised table will be installed and the APPS synonym will be updated to point to the revised table automatically.
  - When a revised temporary table index is delivered with XDF, the revised table will be installed and the APPS synonym will be updated to point to the revised table automatically.
  - Refer to Section 1.4.3.3: Temporary Tables for more information.
- Do not drop a temporary table until the Cleanup phase of the patching process.

  Note: When a revised temporary table is delivered with XDF, the old table will be dropped during cleanup automatically.

Section 2.4.4: Advanced Queue (AQ)

For information on Oracle Streams Advanced Queuing, see: *Oracle Streams Advanced Queuing User's Guide*.

Definition Standards

AQ Payload type must be a non-editioned UDT owned by APPS_NE and referenced by its fully qualified name; for example: `APPS_NE.<type_name>`.

Note: AQ Tables do not get an editioning view. (dbcc: SECTION-27)

Usage Standards

No special considerations.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

Do not alter or drop an existing AQ in a way that is incompatible with the running application.

Code Example: Deferred AQ Drop

```
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=last \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd

-- Example logic to drop an advanced queue during online patching
--
-- To use this logic for another table, you must substitute "&un_fnd" with the
-- actual AQ owner token ("&un_<app_short_name>"), and
-- "FND_EXAMPLE_QUEUE" with the actual AQ name.
--
-- Usage
--    @TABLE_DROP_SCRIPT <table_owner>

drop synonym FND_EXAMPLE_QUEUE;
begin
  ad_zd.load_ddl('CLEANUP',
                 'begin'
                || '  dbms_aqadm.stop_queue(''&&1..FND_EXAMPLE_QUEUE'', FALSE);'
                || '  dbms_aqadm.drop_queue(''&&1..FND_EXAMPLE_QUEUE'');'
                || '  dbms_aqadm.drop_queue_table(''&&1..AQ$_FND_EXAMPLE_QUEUE_TABLE'', TRUE);'
                || 'end;');
end;
/

commit;
exit;
```

Section 2.4.5: Application Context

For information on application contexts, see: *Oracle Database Security Guide*.

Definition Standards

No special considerations.

Usage Standards

No special considerations.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

Do not replace an existing application context definition. You can modify the application context PL/SQL package as needed.

Do not drop an existing application context definition until the cleanup phase of the patching process.

Section 2.4.6: XML Schema

For information on XML Schema and Oracle XML DB, see: *Oracle XML DB Developer's Guide*.

Definition Standards

- The XMLSCHEMA owner must be the "non-editioned APPS" user: `APPS_NE`. Existing XMLSCHEMA types will be automatically upgraded to comply with this standard during the Release 12.2 upgrade.
- Reference the XMLSCHEMA using its fully qualified name: `APPS_NE.<xmlschema_name>`.

Usage Standards

No special considerations.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

Do not alter or drop the XMLSCHEMA definition if it will impact the running application.

- Do not modify an existing XMLSCHEMA; instead create a new XMLSCHEMA.
- Do not drop an existing XMLSCHEMA until the cleanup phase of the patching process.

Section 2.4.7: Database Link

For information on creating database links, see: *Oracle Database Administrator's Guide*.

Definition Standards

Not applicable.

Usage Standards

Accessing objects over a database link always uses the run edition of the target database, even if the access originates from the patch edition.

Dynamic DDL Standards

No special considerations.

Online Patching Compliance Standards

Not applicable.

Section 2.4.8: Partition

Definition Standards

No special considerations.

Usage Standards

No special considerations.

Dynamic DDL Standards

No special considerations.

Section 2.4.9: External Data File

An external data file is user or business data stored on the file system rather than in the database. Examples of external data files include concurrent report output, log files, data import/export files, and file-based system integration. External data files are either human readable or have a structure that is mostly independent of the application code level. These files are not directly patched by Oracle E-Business Suite and (like other types of business data) they should not be editioned.

In Oracle E-Business Suite Release 12.2, the APPL_TOP file system is now editioned, and is no longer suitable for storing external data files. The online patching architecture therefore introduces a new non-editioned file system for the purpose of holding external data files and any other non-editioned files.

Definition Standards

Configure your application to store or access external data files in the non-editioned file system: APPL_TOP_NE. **[minimal]**

- Non-editioned file types include:
    - Report, output, and log files

- Import and export files
- File-based system integration
- Any other data files that are dynamically generated or consumed by the runtime application

- APPL_TOP_NE has the same directory structure as APPL_TOP:
  - `$APPL_TOP_NE/<product_short_name>/<directory_name_used_before>/`
  - Subdirectories are sparsely populated (only created as needed).

Usage Standards

Reference the non-editioned APPL_TOP using the 'APPL_TOP_NE' environment variable.

For example, `$APPL_TOP_NE/fnd/import`

Section 2.5: Additional Information and Examples

Section 2.5.1: Example of Disabling Functionality During Online Patching

Here is a code example for disabling functionality while online patching:

```
--
-- Test if online patching is in progress, if so, block running
--
if ad_zd.get_edition('PATCH') is not null then
  -- an online patch is in progress, return error
  fnd_message.set_name('FND', 'AD_ZD_DISABLED_FEATURE');
  raise_application_error ('-20000', fnd_message.get);
end if;
```

Section 2.5.2: Blocking a Concurrent Program while Online Patching

To block a concurrent program from running while Online Patching is in progress, you must define an incompatibility rule for the concurrent program, as follows:

1. Use the Concurrent Program window or page to edit your concurrent program definition.
2. Select the **Incompatibilities** button to open the Incompatible Programs window.
3. Add a new global incompatibility rule for your program with the following program:
   - Application Name: Applications DBA
   - Program Name: Online Patching In Progress (internal name: ADZDPATCH) concurrent program

Section 2.5.3 Examples of SQL Replacements for PL/SQL Functions

To "edition-enable" the APPS schema, non-editionable objects must not depend on editionable objects. To meet this requirement, the database object development standards specify that materialized views (which are non-editionable) must not call PL/SQL functions (which are editionable).

The examples below demonstrate how to replace frequently-used Oracle Applications Technology PL/SQL function calls with an equivalent SQL in materialized views. You may continue to call built-in PL/SQL functions such as "upper()".

Replacing fnd_profile.value() with an SQL sub-select

Before:

```
fnd_profile.value('MSC_HUB_REGION_INSTANCE')
```

After:

```
(select profile_option_value from fnd_profile_option_values
where level_id = 10001 and (profile_option_id, application_id) =
(select profile_option_id, application_id from fnd_profile_options
where profile_option_name = 'MSC_HUB_REGION_INSTANCE'))
```

Notes:

- This replacement is valid only in a materialized view. For other uses of fnd_profile.value(), continue using the normal PL/SQL call.
- The general case for fetching profile option values is very complex, that is why there is a PL/SQL package dedicated to doing it. But materialized views results have to be valid in any context, so profile options referenced in materialized views should only have site level values, and the replacement SQL only needs to support fetching the site level value.
- This replacement SQL will only use the profile option value set at site level.

Replacing fnd_message.get_string() with an SQL sub-select

Before:

```
fnd_message.get_string('MSC','MSC_HUB_UNASSIGNED')
```

After:

```
(select substrb(REPLACE(message_text, '&&', '&'),1,2000)
from fnd_new_messages m, fnd_application a
where m.message_name = 'MSC_HUB_UNASSIGNED'
and m.language_code = 'US'
and a.application_short_name = 'MSC'
and m.application_id = a.application_id)
```

Notes:

- This replacement is valid only in a materialized view. For other uses of fnd_message.get_string(), continue using the normal PL/SQL call.
- This replacement SQL will only retrieve the US language message text and is not sensitive to any session language settings.
- Materialized view queries cannot contain a sub-SELECT within the main SELECT clause; therefore, the replacement SQL needs to be more sophisticated if the function call was used in the MV SELECT clause.

  Before:

  ```
  select fnd_message.get_string('FND', 'CANCEL')
  from dual
  where 1=1
  /
  ```

  After:

  ```
  select fmgs.result
  from dual
  , (select substrb(REPLACE(message_text, '&&', '&'),1,2000) result
  from fnd_new_messages m, fnd_application a
  where m.message_name = 'CANCEL'
  and m.language_code = 'US'
  and a.application_short_name = 'FND'
  and m.application_id = a.application_id) fmgs
  where 1=1
  /
  ```

Replacing fnd_global.lookup_security_group() with an SQL sub-select

Before:

```
 fnd_global.lookup_security_group('INTEREST_STATUS', 279)
```

After:

```
(select nvl(max(lt.security_group_id), 0)
from fnd_lookup_types lt
where lt.view_application_id = 279
and lt.lookup_type = 'INTEREST_STATUS'
and lt.security_group_id in (
0,
to_number(decode(substrb(userenv('CLIENT_INFO'),55,1),
' ', '0',
null, '0',
 substrb(userenv('CLIENT_INFO'),55,10))))
```

Note: This replacement is valid only in a materialized view. For other uses of fnd_global.security_group(), continue using the normal PL/SQL call.

Section 2.5.4: Example of a Deferred Drop

Here is a code example of a deferred drop. This example is for a table:

```
REM dbdrv: sql ~PROD ~PATH ~FILE \
REM dbdrv:   none none none sqlplus &phase=last \
REM dbdrv:   checkfile:~PROD:~PATH:~FILE &un_fnd
-- Example logic to drop a table under online patching
--
-- Note:  This script drops example index "APPLSYS.FND_EXAMPLE_U1" and
-- table "APPLSYS.FND_EXAMPLE_TABLE".   An unwanted unique index must
-- dropped during CUTOVER phase.  All other unwanted non-editioned objects
-- can be dropped during the CLEANUP phase.
--
-- Note: If there are DDL/ DML statements, which have running-order-dependency
--       within a PHASE, then use a PL/SQL procedure or an anonymous pl/sql
--       block to take care of the running order. For example:

--       DROP INDEX "index-name" and CREATE INDEX "index-name" individual DDL statements,
--       populated for "CUTOVER" phase, are having order-dependency, which would be a problem
--       in parallel-running, so such individual DDL statements should be put together
--       within a PL/SQL API.
--
--
-- To use this logic for another table, you must substitute "&un_fnd" with the
-- actual table owner token ("&un_<app_short_name>"), and
-- "FND_EXAMPLE_TABLE" with the actual table name.
--
-- Usage
--    @TABLE_DROP_SCRIPT <table_owner>
drop synonym FND_EXAMPLE_TABLE;
drop view &1..FND_EXAMPLE_TABLE#;
exec ad_zd.load_ddl('CUTOVER', 'drop index &1..FND_EXAMPLE_TABLE_U1')
exec ad_zd.load_ddl('CLEANUP', 'drop table &1..FND_EXAMPLE_TABLE')
```

Part 3: Database Object Development Standards for Oracle E-Business Suite System Schema Migration

Section 3.1: Introduction

This section documents new or modified standards for database object development and the Oracle E-Business Suite System Schema Migration in Oracle E-Business Suite Release 12.2. This section only covers standards that are required for recommendations and guidelines in support of the Oracle E-Business Suite System Schema Migration; it does not integrate other general development standards.

Recommended Guideline

The majority of the development standards documented for the Oracle E-Business Suite System Schema Migration are recommended guidelines for your customizations and are not mandatory. Many of the development standards are not new but rather have been long-standing recommended Oracle E-Business Suite or Oracle Database standards. Migrating customizations to later Oracle database releases and Oracle database platforms will be easier if you follow the recommended guidelines.

If you do not modify custom code that falls into the category of a recommended guideline, then the custom code should continue to function without error. Exceptions to this include requirements mandated by later releases of the Oracle database or by your Oracle database platform. For example, upgrading to a later database release may require the recommended guideline.

Also note that all of the recommended standards in this document are mandatory requirements for migrating your Oracle E-Business Suite database to Oracle Autonomous Database. Refer to the Oracle Autonomous Database documentation at https://docs.oracle.com/en/cloud/paas/autonomous-database/index.html for more information.

Mandatory Compliance

For the mandatory compliance items, you must update the identified custom-defined objects that reference Oracle E-Business Suite code that previously resided in the SYS or SYSTEM schemas prior to applying the Oracle E-Business Suite System Schema Migration Completion Patch (referred to as simply the Completion Patch). It is estimated that the number of customizations with such references is low. The mandatory update is to change any references from EBS-defined object that resided in either the SYS schema or SYSTEM schema to the migrated EBS-defined object in either the APPS schema or APPS_NE schema.

Warning: If you do not update the identified custom code objects that have mandatory compliance requirements, your custom code will have invalid database objects and will run with errors after applying the Completion Patch.

Section 3.2: Development Standards

This section includes EBS System Schema Migration development standards which are either recommended guidelines or mandatory compliance requirements for custom-defined objects. Mandatory compliance requirements must be resolved so that your customizations will run without error after applying the Completion Patch. Failure to do so will result in invalid database objects for your custom code. To review the results compliance report, run the Oracle E-Business Suite System Schema Migration Compliance Checker (ADSYSCC.sql) as follows:

```
sqlplus <APPS username> @$AD_TOP/sql/ADSYSCC.sql
```

If the results of the ADSYSCC.sql returns any rows that include custom-defined objects, then the custom code is not deployed according to Oracle development recommended guidelines and/or mandatory requirements. Refer to the following sections for recommended guidelines and mandatory requirements for custom-defined code.

3.2.1: Custom-Defined Database Object in Oracle Database System Accounts

Definition Standards

Custom-Defined database objects should not reside in Oracle database system accounts (e.g., SYS, SYSTEM, CTXSYS). (syscc: SECTION-1.1)

Recommended Guideline

We recommend that custom-defined objects are migrated according to the guidance in the previous sections.

3.2.2: References to Oracle E-Business Suite-Defined or Custom-Defined Objects in Database System Accounts (Required)

Definition Standards

Custom-defined objects that reference custom-defined objects that reside in the Oracle Database system accounts should reference migrated objects. (syscc: SECTION-1.2)

Recommended Guideline

We recommend that you migrate any custom-defined database objects according to the development standards documented in Part 1: Developing and Deploying Customizations in Oracle E-Business Suite Release 12.2 and Part 2: Database Object Development Standards for Online Patching of this note.

After the custom-defined objects are migrated, it is recommended to update the custom-defined object to reference the code that has been migrated per the documented development standard.

Mandatory Compliance Requirement

- It is mandatory that custom-defined database objects that reference EBS-defined database objects that reside in Oracle Database system accounts (e.g., SYS, SYSTEM, CTXSYS) be updated to reference the EBS-defined objects once migrated to EBS-defined schemas.
- Migration of EBS-defined objects that reside in Oracle Database system accounts occurs with the application of AD Delta 13, TXK Delta 13 and the EBS System Schema Migration Consolidated patches.

Warning: Prior to running the Oracle E-Business Suite System Schema Migration Completion Patch, all Oracle E-Business Suite code will have been migrated from the SYS or SYSTEM schemas to the APPS or APPS_NE schemas. The Oracle E-Business Suite System Schema Migration Completion Patch will drop all Oracle E-Business Suite code that resides in the SYS or SYSTEM accounts to finalize the migration process.

Before you run the Oracle E-Business Suite System Schema Migration Completion Patch all custom code that points to EBS-defined objects in Oracle Database System Accounts must be updated to point to the migrated EBS-defined objects in EBS-defined schemas. Failure to due so will result in errors with your custom code. For more information, refer to My Oracle Support Knowledge Document 2755875.1, *Oracle E-Business Suite Release 12.2 System Schema Migration*.

3.2.3 Custom-Defined Type Objects Referenced as Table Columns in the SYS or SYSTEM Schemas

Definition Standards

Custom-defined TYPE objects that are referenced as table columns and reside in the SYS or SYSTEM schemas should be migrated. (syscc: SECTION-1.3)

Recommended Guideline

We recommend that custom-defined TYPEs are migrated to the APPS_NE schema per the standards documented in 2.4.2 User-Defined Type (UDT) (Non-Editioned).

To do so, perform these steps:

1. Create the custom defined type referenced as table column in the APPS_NE schema.
2. Grant the privileges using the API as follows:

   ```
   AD_SEC_UTILS.GRANT_PRIVS_ON_APPS_NE_TYPE
   ```

   For example:

   ```
   begin
       AD_SEC_UTILS.GRANT_PRIVS_ON_APPS_NE_TYPE('CCT_IBMEDIA_TYPE');
         end;
   ```

3. Create a synonym in the APPS schema to the `APPS_NE.<UDT>` created in step 1.
4. Fix all Table Column or AQ references from `SYSTEM.<UDT>` to `APPS_NE.<UDT>`.

   You can do this in one of two ways:

- With an online patch using the cutover DDL

```
begin
    ad_zd.load_ddl('CUTOVER',
        'begin
     AD_SEC_UTILS.FIX_SYSTEM_OWNED_NE_TYPE( ''CCT_IBMEDIA_TYPE'' );
         end;');
end;
```

or,

- With downtime using this package call:

```
begin
   AD_SEC_UTILS.FIX_SYSTEM_OWNED_NE_TYPE( ''CCT_IBMEDIA_TYPE'' );
end;
```

The above API is used to fix the type. You can do this in an Online Patching cycle (change in effect after cutover) or run the API in non-production or with downtime in production.

3.2.4 Custom-Defined Type Objects Not Referenced as Table Columns in the SYS or SYSTEM Schemas

Definition Standards

Custom-defined TYPE objects that are not referenced as table columns and reside in the SYS or SYSTEM schemas should be migrated. (syscc: SECTION-1.4)

Recommended Guideline

We recommend that you migrate custom-defined TYPE objects according to the standards documented in 2.2.4 User-Defined Type (Editioned).

Use the following steps to migrate your objects.

1. Create the custom-defined type not referenced as table columns in either a custom schema or the APPS schema.

   The definition can be extracted using the xdfgen.pl utility as follows:

   ```
   perl xdfgen.pl <apps_user>/<apps_password>@<DB_SID> <OBJECT_NAME> OBJECT_TYPE=TYPE OWNER_APP_SHORTNAME=<APP_SHORT_NAME>
   ```

2. Drop the custom-defined type from SYSTEM.

3.2.5 References to Oracle E-Business Suite-Defined or Custom-Defined TYPE Objects in Database System Accounts (Required)

Definition Standards

Custom-defined objects with references to custom-defined TYPE objects that reside in Oracle Database system accounts (e.g., SYS, SYSTEM, CTXSYS) should reference migrated objects. (syscc: SECTION-1.5)

Recommended Guideline

We recommend that you migrate custom-defined database TYPE objects according to the documented standards in 2.2.4 User-Defined Type (Editioned) or 2.4.2 User-Defined Type (UDT) (Non-Editioned).

After the custom-defined database object is migrated, you should update the custom-defined object to reference the code that has been migrated.

Mandatory Compliance Requirement

It is a mandatory requirement that custom-defined database objects that reference EBS-defined TYPE database objects that reside in Oracle Database system accounts (e.g., SYS, SYSTEM, CTXSYS) be updated to reference the EBS-defined TYPE objects once migrated to EBS-defined schemas

Migration of the EBS-defined TYPE database objects that reside in Oracle Database system accounts occurs with the application of AD Delta 13, TXK Delta 13 and the EBS System Schema Migration Consolidated patches.

Warning

Prior to running the Oracle E-Business Suite System Schema Migration Completion Patch, all Oracle E-Business Suite code will have been migrated from the SYS or SYSTEM schemas to the APPS or APPS_NE schemas. The Oracle E-Business Suite System Schema Migration Completion Patch will drop all Oracle E-Business Suite code that resides in the SYS or SYSTEM accounts to finalize the migration process.

Before you run the Oracle E-Business Suite System Schema Migration Completion Patch all custom code that points to EBS-defined objects in Oracle Database System Accounts must be updated to point to the migrated EBS-defined objects in EBS-defined schemas. Failure to do so will result in errors with your custom code. For more information, refer to My Oracle Support Knowledge Document 2755875.1, *Oracle E-Business Suite Release 12.2 System Schema Migration*.

3.2.6 Custom-Defined Synonyms that Resolve to Objects in Oracle Database System Accounts

Definition Standards

Custom-defined synonyms that resolve to custom-defined objects in Oracle Database System Accounts should reference migrated objects. (syscc: SECTION-1.6)

Recommended Guideline

We recommend that the referenced custom-defined objects that reside in the Oracle Database system accounts should be migrated per the documented standards in 2.2.4: User-Defined Type (Editioned) or 2.4.2: User-Defined Type (UDT) (Non-Editioned).

After the custom-defined object is migrated, update the custom-defined synonym to reference the migrated object.

Mandatory Compliance Requirement

It is a mandatory requirement that custom-defined synonyms that reference EBS-defined database objects that reside in Oracle Database system accounts (e.g., SYS, SYSTEM, CTXSYS) must be updated to reference the EBS-defined objects once migrated to EBS-defined schemas.

Migration of the EBS-defined TYPE database objects that reside in Oracle Database system accounts occurs with the application of AD Delta 13, TXK Delta 13 and the EBS System Schema Migration Consolidated patches.

Warning

Prior to running the Oracle E-Business Suite System Schema Migration Completion Patch, all Oracle E-Business Suite code will have been migrated from the SYS or SYSTEM schemas to the APPS or APPS_NE schemas. The Oracle E-Business Suite System Schema Migration Completion Patch will drop all Oracle E-Business Suite code that resides in the SYS or SYSTEM accounts to finalize the migration process.

Before you run the Oracle E-Business Suite System Schema Migration Completion Patch all custom code that points to EBS-defined objects in Oracle Database System Accounts must be updated to point to the migrated EBS-defined objects in EBS-defined schemas. Failure to do so will result in errors with your custom code. For more information, refer to My Oracle Support Knowledge Document 2755875.1, *Oracle E-Business Suite Release 12.2 System Schema Migration*.

3.2.7 Custom-Defined Code that References Private SYS.DBMS_* Packages

Definition Standards

Oracle Database coding standards state that utilize known private internal SYS-owned packages is not recommended. (syscc: SECTION-1.7)

Recommended Guideline

We recommend that custom-defined objects that utilize known private internal SYS-owned packages be updated to reference public APIs.

For example, Oracle E-Business Suite has updated its code as follows:

- References to DBMS_PIPE for inter-process communication were updated as DBMS_PIPE should only be used for logging or tracing and not for inter-process communications
- References to DBMS_SYSTEM, DBMS_REGISTRY, DBMS_METADATA_UTL, DBMS_REPCAT, DBMS_SCHEMA_COPY, DBMS_OBJECTS_APPS_UTILS, DBMS_PRVTAQIM, DBMS_AQADM_SYS were removed.

Refer to the Oracle Database documentation for more information regarding public Oracle database APIs that support your required custom functionality.

3.2.8 Custom-Defined Code that References Private SYS Objects

Definition Standards

Oracle Database coding standards state that queries that utilize known private internal SYS-owned data dictionary tables or views are not recommended. (syscc: SECTION-1.7)

Recommended Guideline

We recommend that custom-defined objects that utlize known private internal SYS owned data dictionary tables or views are updated to reference public APIs. For example:

- References to OBJ$ should be updated to reference ALL_OBJECTS
- References to USER$ should be updated to reference ALL_USERS.

Refer to the Oracle Database documentation for more information regarding public Oracle database APIs that support your required custom functionality.

3.2.9 Desupport/Deprecation of Oracle Multimedia

Definition Standards

Oracle Database guidelines state that you should store multimedia content in SecureFiles LOBs, and use open source or third-party products such as Piction for image processing and conversion. (syscc: SECTION-2.1)

Note: Oracle Multimedia is deprecated with Oracle Database 18c. Oracle Locator is not affected by the deprecation of Oracle Multimedia.

Recommended Guideline

It is recommended to remove custom-defined dependencies on Oracle Multimedia.

Refer to the Oracle Database administration and development documentation for guidance.

3.2.10 Tables with XMLType Column Storage Violations

Definition Standards

Custom-defined XMLType Columns with unstructured storage defined as CLOB or OBJECT-RELATIONAL are deprecated with Oracle Database 12c (syscc: SECTION-2.2),

Recommended Guideline

We recommend that custom-defined tables with XMLType columns that use CLOB or OBJECT-RELATIONAL are updated to use binary XML storage of XMLType.

Refer to the Oracle Database documentation for more information on converting CLOB or OBJECT-RELATIONAL storage of XMLType to binary XML.

To update the tables, perform these steps:

1. Create a revised column for the XMLTYPE columns.
2. Follow the instructions in Section 1.4.3.1: Tables to update an existing column.

3.2.11 XMLType Tables with Storage Violations

Definition Standards

Custom-defined XMLType tables with unstructured storage defined as CLOB or OBJECT-RELATIONAL are deprecated with Oracle Database 12c, (syscc: SECTION-2.3)

Recommended Guideline

We recommend that custom-defined XMLType tables with CLOB or OBJECT-RELATIONAL storage are updated to use binary XML storage of XMLType.

Refer to the Oracle Database documentation for more information on converting CLOB or OBJECT-RELATIONAL storage of XMLType to binary XML.

To update a table, perform the following steps:

1. Create a revised column for the XMLTYPE column.

   For example:

   ```
   ALTER TABLE ego.ego_odi_ws_xsl_zd ADD (
       "XSLCONTENT#1" sys.xmltype
   ) XMLTYPE COLUMN "XSLCONTENT#1" STORE AS BINARY XML;
   ```

2. Follow the instructions in Section 1.4.3.1: Tables to update an existing column.

3.2.12 XMLType Schemas

Definition Standards

Subprograms in DBMS_XMLSCHEMA, many DBMS_XDB subprograms and many other Oracle XML DB schema features are desupported as of Oracle Database 18c. (syscc: SECTION-2.4)

Recommended Guideline

We recommend that custom-defined objects in XMLTYPE SCHEMAS are upgraded to remove the dependencies and to drop the XML Schemas.

Refer to the Oracle Database documentation for more information on dropping XML Schemas.

Use the API described below to fix the type. You can do this in an Online Patching cycle (change in effect after cutover), run the API in a non-production instance, or run the API with downtime in a production instance.

You can drop the XML Schema in one of two ways:

- Submit a deferred DDL statement to remove the XML schemas during the adop phase `XML_SCHEMA_CLEANUP`. For example:

  ```
  exec ad_zd.load_ddl ('XML_SCHEMA_CLEANUP','BEGIN AD_SEC_UTILS.REMOVE_XML_SCHEMA(X_SCHEMA_URL=>''http://isetup.example.com/2006/diffresultdata.xsd''); end; ' )
  ```

- Alternatively, call the API AD_SEC_UTILS.REMOVE_XML_SCHEMA to drop the XML Schema.

  For example:

  ```
  BEGIN
  AD_SEC_UTILS.REMOVE_XML_SCHEMA(X_SCHEMA_URL=>''http://isetup.example.com/2006/diffresultdata.xsd''); end;
  ```

Appendix: Additional Information

Working with Customizations using Online Patching

Customers who have Oracle E-Business Suite Release 12.2 customization but do not want to use Online Patching when changing their own customizations can do so using traditional "downtime patching" and manual steps.

Customers doing so need to keep the following considerations in mind:

- Custom objects that reside in the database and reference Oracle E-Business Suite objects must reside in an editioned schema.
- Custom objects that reside in the database and in the application tier that access the data model must access it through the synonyms in the APPS schema (logical layer).
- Changes to custom code that resides in the file system (for example, Forms, Oracle Application Framework code, Java, and so on) have to be kept in sync between the 2 file systems. The recommended approach for keeping them in sync is by using the Custom Synchronization Driver.
- Changes to custom code that resides in the database have two alternatives. The alternatives depend on the fact that an Online Patching is underway or not.
  - If a Patching cycle is underway, the changes have to be applied to the Patch edition in the database; in this case the changes won't be visible to the end users until the administrator performs a cutover of the patching cycle that is underway.
  - If there is no Patching Cycle underway, the administrator can apply the customization changes to the Run edition of the database directly; in this case the recommended approach is to use a downtime and apply the changes.

To patch custom code using downtime patching when there is no online patching cycle in progress

1. Stop middle-tier application services:

   ```
   $ source <ebs_root>/EBSapps.env run
   $ adstpall.sh
   ```

2. Apply custom patches to the run edition of the file system and database:

   ```
   $ source <ebs_root>/EBSapps.env run
   $ ... apply custom patches ....
   ```

3. Start middle-tier application services:

   ```
   $ adstrtal.sh
   ```

To patch custom code using downtime patching when there is an online patching cycle in progress:

1. Wait for the online patching cycle to complete the cutover phase. When running the online patching cutover, give the `mtrestart=no` option to prevent application servers from restarting after the cutover:

   ```
   $ adop phase=cutover mtrestart=no
   ```

2. Apply custom patches to the new run edition of the file system and database:

   ```
   $ source <ebs_root>/EBSapps.env run
   $ ... apply custom patches ....
   ```

3. Start middle-tier application services:

   ```
   $ adstrtal.sh
   ```

4. Complete the online patching cycle:

   ```
   $ adop phase=cleanup
   ```

Change Log

Date	Description
12 Dec 2023	• Added additional step to Section 1.5.3 if customall.jar is a client custom jar file.
03 Feb 2022	• Formatting and minor textual changes.
11 Nov 2021	• Added information on scripts delivered in R12.AD.C.Delta.13 to Section 1.1.4. • Added Part 3, "Database Object Development Standards for Oracle E-Business Suite System Schema Migration." • Formatting and minor textual changes.
29 Jan 2018	• Added information on working with custom materialized views in Oracle Database 12c to Section 1.4.3.5, "Materialized Views." • Updated syntax of command "`rsync -zr %s_current_base%/EBSapps/comn/java/classes/<Company identifier>/` ..." in Section 1.5.4, "Adding Entries to the Custom Synchronization Driver File."
30 Oct 2017	• Updated syntax for xdfcmp.pl command in Section 1.4. • Updated references to other documents.
04 Nov 2016	• Updated section 1.4.3.1, "Tables," regarding file versions; updated subsection "Migrate Data to a New Table." • Updated section 1.4.3.5, "Materialized Views," to account for new Oracle Database 12.1. • Added a new section 2.2.1, "Granting Privileges to Editioned Objects," describing the API AD_ZD.GRANT_PRIVS. • Updated section 2.2.2, "View (Ordinary)," to clarify the standard for a join view defining a ROW_ID column mapped to the ROWID of the base table. • Updated document title for My Oracle Support Knowledge Document ID 1927975.1, *Oracle E-Business Suite Release 12.2 Upgrade Considerations for OAF-based Applications and Oracle CRM Technology Foundation*.
15 Sep 2016	Updated section 1.4, "Developing and Deploying Custom Code and Custom Database Objects," with new syntax for xdfgen.pl and xdfcmp.pl in R12.AD.C.Delta.8.
01 Jun 2016	Updated section 1.1.2, "Connecting to the Patch Edition," with a note on using command line tools.
11 Apr 2016	Updated section 2.3.1, "Table (Ordinary)," regarding how to drop a logical column. Updated section 2.3.6, "Materialized View (MV)," in the code example (removed the comment '`/*AUTOREFRESH*/`'). Updated section 2.4.4, "Advanced Queue (AQ)," with a code example for how to drop an advanced queue during online patching.
23 Mar 2016	Updated section 1.6.6, "Oracle Workflow," with a new subsection "1.6.6.3, Customizing Approvals Data Services Configuration Metadata." Clarified section 2.2 for Dynamic DDL Standards for views, PL/SQL packages, PL/SQL triggers, and synonyms.
29 Jan 2016	Updated Section 1.4, to add in a new subsection 1.4.1, "Recommended Approaches For Deploying Custom Code." Updated Part 2, Database Object Development Standards: • Changed the [run] tag to [minimal] for Part 2, Database Object Development Standards. • Updated Ordinary Tables with the requirement that new columns with a default value must be not null. • Updated Seed Data Tables section with prohibition on disabling the ZD_SEED VPD policy or generated maintenance trigger. Updated Part 1, Developing and Deploying Customizations in Oracle E-Business Suite Release 12.2: • Section 1.1, "Working with Editions": Updated the descriptions for ADZDALLDDLS, ADZDSHOWIOBJS, and ADZDSHOWCOBJS. • Section 1.2, "Applying Online Patches": Added step for validating the system under the Prepare phase. Under the Apply phase, for Oracle patches, changed the directory for ZIP files to "$PATCH_TOP". Also under that subsection, added note regarding applying replacement patches and using the "abandon=yes" parameter. For applying manual patches, added step to execute commands to generator or compile dependent files or objects. • Section 1.2, "Applying Online Patches": Added descriptions for FS Clone and Abort under 1.2.7, "Special Patching Actions." • Section 1.3, "Developing Customizations": Updated introduction. • Section 1.4.1, "Editioned Database Objects": Under Step 1, added a note for confirming that there are no unexpected invalid objtects. • Section 1.4.2.4, "Temporary Tables": Under "Revise an Existing Temporary Table, revised instructions to run xdfgen.pl on the APPS synonym for the temporary table. Also added note regarding ensuring that a temporary table with a name matching the APPS synonym exists in addition to the revised temporary table to be extracted.
27 Oct 2015	Updated Part 2, Database Object Development Standards with section "Full Compliance versus Runtime Compliance" and related **[run]** tags; and additional updates.
08 Oct 2015	Updated syntax for xdfgen.pl for R12.AD.C.Delta.7 (Sections 1.4.2.1 to 1.4.2.7).
17 Mar 2015	• Updated syntax for xdfgen.pl (Sections 1.4.2.1 to 1.4.2.7). • Added Section 1.4.3 "Advanced Topics," for Data Fixer Patch and Partition Exchange. • Updated Section 1.4.2.1 "Effectively-Editioned Database Objects: Tables" (Update an Existing Column, and Migrate Data to a New Table). • Updated Section 1.4.2.4, "Effectively-Editioned Database Objects: Temporary Tables" (Revise an Existing Temporary Table). • Updated Section 1.2.6 regarding cleanup modes. Added update for customizing ebsProductManifest_xml.tmp for R12.TXK.C.DELTA.6 (Section 1.5.5, step 4). • Added Part 2, Database Object Development Standards.
06 Jan 2015	Added reference to *Guidance for Integrating Custom and Third-Party Products With Oracle E-Business Suite Release 12.2* (My Oracle Support Knowledge Document 1916149.1).
31 Oct 2014	Added reference to *Oracle E-Business Suite Release 12.2.4 Upgrade Considerations for OA Framework-based Applications* (My Oracle Support Knowledge Document 1927975.1).
30 Oct 2014	Updated paragraph under "Form Libraries (.pll and .plx files)" in Section 6.2.3, "Deploying Forms Files": After running cutover, ensure that synchronization commands for custom files are defined in the custom synchronization driver.
06 Oct 2014	Updated paragraph in Section 6.2.3, "Deploying Forms Files."
05 Sep 2014	Made minor edits to Section 5.5, "Deploying Java Files at Non-Standard Location(s) for Custom Products."
11 Aug 2014	Updated title.
20 Mar 2014	Updated references for 's_run_base' and 's_patch_base' to 's_current_base' and 's_other_base', respectively. Added additional information for XMLImporter to Section 6.3.
06 Mar 2014	Added references in Section 6.3 to Document 1315485.1 for the *Oracle Application Framework Developer's Guide*.
13 Dec 2013	Added Section 5.5, "Deploying Java Files at Non-Standard Location(s) for Custom Products." Updated note for Release 12.2.3, including notes regarding abort and full cleanup in adop. Updated formatting.
20 Sep 2013	Corrected name (capitalization) of 'ebsProductManifest.xml' in Section 5.3, "Running the adcgnjar Utility."
19 Sep 2013	Initial publication.

REFERENCES

Attachments

Developing and Deploying Customizations in Oracle E-Business Suite Release 12.2 (500.58 KB)

Related

Products

- Oracle E-Business Suite > Applications Technology > Integration > Oracle EDI Gateway > Receivables > EDI Invoice
- Oracle E-Business Suite > Applications Technology > Application Object Library > Oracle Concurrent Processing > MICC Transfer to TSC
- Oracle E-Business Suite > Applications Technology > Lifecycle Management > Oracle Applications DBA > Online Patching - OP
- Oracle E-Business Suite > Applications Technology > Technology Components > Oracle E-Business Suite Technology Stack > IAS for Applications Technology > IAS for Applications Technology
- Oracle E-Business Suite > Applications Technology > Application Object Library > Oracle Application Object Library > Basic SysAdmin functions, maintenance > ""Auditing""

Keywords

ADOP; COMPLIANCE; CONCURRENT PROGRAMS; CROSS-EDITION TRIGGER; CUSTOM; DOCUMENTATION; E-BUSINESS; EBS; MATERIALIZED VIEW; UPGRADE; WORKFLOW

Errors

06512; CONC-MISSING REQ FILE BASEPATH; ORA-00923; ORA-01446; ORA-38818

Translations

- English Source
- Japanese 日本語

Back to Top