Oracle® Configurator

Implementation Guide

Release 12.2

Part No. E48816-01

September 2013

ORACLE

Oracle Configurator Implementation Guide , Release 12.2

Part No. E48816-01

Primary Author: Margot Murray

Contributing Author: Tom Myers

Contents

2 Configurator Architecture

3 Database Instances

Part 2 Data

4 The CZ Schema

5 Populating the CZ Schema

6 Migrating Data

7 Synchronizing Data

8 CZ Schema Maintenance

Part 3 Integration

9 Session Initialization

10 Session Termination

11 Batch Validation

12 Custom Integration

13 Pricing and ATP in Oracle Configurator

14 Multiple Language Support

Part 4 Configuration Model

15 Controlling the Development Environment

16 Publishing Configuration Models

17 Programmatic Tools for Development

18 Programmatic Tools for Maintenance

Part 5 Runtime Configurator

19 User Interface Deployment

20 Deployment Considerations

21 Managing Configurations

A Terminology

B Common Tasks

C Concurrent Programs

D CZ Subschemas

E Code Examples

Common Glossary for Oracle Configurator

Index

Send Us Your Comments

Oracle Configurator Implementation Guide , Release 12.2

Part No. E48816-01

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document. Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Oracle E-Business Suite Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: appsdoc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

Intended Audience

Welcome to Release 12.2 of the *Oracle Configurator Implementation Guide* .

This guide presents tasks and information useful in implementing Oracle Configurator.

See the *Oracle Configurator Installation Guide* for installation information, the *Oracle Configurator Developer User's Guide* for information about developing configuration models in Oracle Configurator Developer, the *Oracle Configurator Modeling Guide* for information about designing configuration models that are best suited to Oracle Configurator, *Oracle Configurator Methodologies* for information and tasks useful in implementing Oracle Configurator, the *Oracle Configurator Extensions and Interface Object Developer's Guide* for information about writing Configurator Extensions, the *Oracle Configurator Constraint Definition Language Guide* for information about writing Statement Rules, and the *Oracle Configurator Performance Guide* for information needed for optimizing runtime performance of Oracle Configurator.

This guide is intended for anyone responsible for supporting the use of Oracle Configurator. This includes supporting the development environment (Oracle Configurator Developer) as well as the runtime environment that is created for deployment.

Ordinarily, the tasks presented in this book are performed by a Database Administrator (DBA) or an Oracle Configurator implementer with DBA experience.

See Related Information Sources on page xxx for more Oracle E-Business Suite product information.

Documentation Accessibility

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

Access to Oracle Support

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

Structure

1 Implementation Tasks

This chapter presents an overview of all known tasks in an Oracle Configurator implementation, including custom tasks.

2 Configurator Architecture

This chapter describes the elements of the Oracle Configurator product and how they fit together.

3 Database Instances

This chapter describes the uses to which databases are put when implementing Oracle Configurator, and specifics about using multiple database instances.

4 The CZ Schema

This chapter describes the basic characteristics of the CZ schema, the schema settings and how they are used, and provides some schema maintenance tips.

5 Populating the CZ Schema

This chapter provides an overview of why and how to import data from Oracle Applications and non-Oracle Applications databases. It describes the import processes, the import tables used during data import, how to import data into the CZ schema, data import verification, the process for refreshing or updating imported data, and customizing data import.

6 Migrating Data

This chapter describes how to migrate a CZ Release 12 instance into an empty CZ instance, and how to migrate Model data from one instance to another development instance.

7 Synchronizing Data

This chapter describes when and how data is synchronized. This includes synchronizing BOM data after the import server has changed and synchronizing publication data after a database has been cloned.

8 CZ Schema Maintenance

This chapter explains how to maintain data when it exists in more than one place and is potentially unsynchronized.

9 Session Initialization

This chapter describes the format and parameters of the initialization message for the runtime Oracle Configurator.

10 Session Termination

This chapter describes the format and parameters of the termination message for the runtime Oracle Configurator Servlet.

11 Batch Validation

This chapter describes using Oracle Configurator in a programmatic mode.

12 Custom Integration

This chapter explains how to modify certain Oracle Configurator files as well as the purpose of the files and where they can be found.

13 Pricing and ATP in Oracle Configurator

This chapter provides an overview of how pricing works in a runtime Oracle Configurator.

14 Multiple Language Support

This chapter explains how Item descriptions are entered in Oracle Applications and can be displayed in multiple languages when deploying an Oracle Configurator User Interface.

15 Controlling the Development Environment

16 Publishing Configuration Models

This chapter explains the database processes for publishing configuration models to make them available to host applications.

17 Programmatic Tools for Development

This chapter describes a set of programmatic tools (PL/SQL procedures and functions) that may be useful in developing a configuration model and deploying a runtime Oracle Configurator.

18 Programmatic Tools for Maintenance

This chapter describes a set of programmatic tools (PL/SQL procedures) that you can use primarily to maintain a deployed runtime Oracle Configurator.

19 User Interface Deployment

This chapter describes the activities required to complete the User Interface deployment of a runtime Oracle Configurator embedded in a host Oracle Application such as Order Management or *i*Store.

20 Deployment Considerations

This chapter describes the strategies you should consider when you are ready to complete the deployment of a runtime Oracle Configurator.

21 Managing Configurations

This chapter describes the data structures produced by Oracle Configurator during a configuration session, and how to manage the life cycle of a configuration.

A Terminology

This appendix defines the terms that are found in the *Oracle Configurator Implementation Guide* that are not defined in the Glossary.

B Common Tasks

This appendix describes certain tasks that may be required while implementing an Oracle Configurator.

C Concurrent Programs

This appendix describes the concurrent programs available to either the Oracle Configurator Administrator or Oracle Configurator Developer responsibility.

D CZ Subschemas

This appendix lists the CZ tables that make up each of the subschemas in the CZ schema. For table details, see the Oracle Integration Repository.

E Code Examples

Common Glossary for Oracle Configurator

Related Information Sources

> **Important:** There is new functionality available for the Runtime Oracle Configurator when using the Fusion Configurator Engine (FCE). The FCE is an alternative to the configuration engine described in this document. For all information about the FCE, see the *Oracle Configurator Fusion Configurator Engine Guide*.

For more information, see the following resources:

- Be sure you are familiar with the latest release or patch information for Oracle Configurator see the Oracle Support Web site.
- For a full list of documentation resources for Oracle Configurator, see the Oracle Configurator Release Notes for this release.
- For a full list of documentation for Oracle Applications, see Oracle Applications Documentation, on the Oracle Technology Network.
- For detailed reference information about the tables in the CZ schema, see the Oracle Integration Repository.
- For useful background in implementing applications, consult the Oracle database documentation resources for the current guidelines on performance methods.

Integration Repository

The Oracle Integration Repository is a compilation of information about the service endpoints exposed by the Oracle E-Business Suite of applications. It provides a complete catalog of Oracle E-Business Suite's business service interfaces. The tool lets users easily discover and deploy the appropriate business service interface for integration with any system, application, or business partner.

The Oracle Integration Repository is shipped as part of the E-Business Suite. As your instance is patched, the repository is automatically updated with content appropriate for the precise revisions of interfaces in your environment.

You can navigate to the Oracle Integration Repository through Oracle E-Business Suite

Integrated SOA Gateway.

Do Not Use Database Tools to Modify Oracle E-Business Suite Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle E-Business Suite data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle E-Business Suite data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle E-Business Suite tables are interrelated, any change you make using an Oracle E-Business Suite form can update many tables at once. But when you modify Oracle E-Business Suite data using anything other than Oracle E-Business Suite, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle E-Business Suite.

When you use Oracle E-Business Suite to modify your data, Oracle E-Business Suite automatically checks that your changes are valid. Oracle E-Business Suite also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

Part 1

Introduction

Part 1 consists of chapters that present a baseline for understanding Oracle Configurator.

1

Implementation Tasks

This chapter presents an overview of all known tasks in an Oracle Configurator implementation, including custom tasks.

This chapter covers the following topics:

- Overview
- General Implementation Tasks
- Database Tasks
- Integration Tasks
- Model Development Tasks
- Deployment Tasks
- Conventions
- Product Support

Overview

This chapter provides an overview of tasks performed prior to implementing Oracle Configurator. The list of tasks is organized into the following categories:

General Implementation Tasks

General implementation tasks are the initial tasks that set up an environment and enable the implementer to begin working with Oracle Configurator Developer.

- Verify Oracle Rapid Install of Oracle Configurator, Oracle Configurator Developer and the CZ schema. See the *Oracle Configurator Installation Guide* for additional information.

- Configure Oracle Configurator Developer, JInitiator, and your browser to display appropriate fonts for Multiple Language Support (MLS). See the *Oracle Configurator Installation Guide* for details.

- See the current release or patch information for Oracle Configurator on Oracle Support Web site, for any effects an Oracle Configurator upgrade may have on your development and test environments; new functionality in Configurator Developer may depend on other applications.

- Upgrade Oracle Configurator Developer to the latest release or patch level. For more information, see the current release or patch information for Oracle Configurator on Oracle Support Web site.

- Assign users an Oracle Configurator responsibility to use Oracle Configurator Developer. For more information about assigning responsibilities, see Setting up Access to Configurator Developer, page 15-2 and the *Oracle E-Business Suite System Administrator's Guide*.

- Assign users either the Oracle Configurator Administrator or Oracle Configurator Developer responsibility to run the Oracle Applications concurrent programs. For more information about assigning responsibilities, see the *Oracle E-Business Suite System Administrator's Guide*. For more information on which concurrent program can be run by the Oracle Configurator Administrator or Oracle Configurator Developer responsibilities, see Concurrent Programs, page C-1.

Database Tasks

Database tasks are the tasks that set up and support the development and deployment of the CZ schema.

Required Database Tasks

These tasks must be performed to set up and support development and deployment of a runtime Oracle Configurator.

- Decide whether to use a single database instance for both development and

production, or a separate instance for development and an instance for production. For information see Database Instances, page 3-1 .

- Verify that Inventory and **BOM** Model data in Oracle Applications are correctly defined. See Standard Import., page 5-4

- Populate the CZ schema with production BOM and Inventory data for use in defining configuration models. This is also referred to as data import in Oracle Configurator documentation. For information, see Standard Import., page 5-4

- Control the scope of the data import by modifying values in the integration tables (CZ_XFR_) provided for that purpose. For information, see Control Tables., page 4-9

- Define and enable servers, as needed for data import, synchronization, and publication. For information, see Server Administration, page B-3 concurrent program.

- Modify Configurator Parameters. The Oracle Configurator Administrator runs this concurrent program to set installation-wide customizable settings (CZ_DB_SETTINGS) that describe the structure and content of the CZ schema, and define application functions. For information, see Modify Configurator Parameters, page C-3 concurrent program.

- Explode the BOM Model data if the data on which you plan to base your configuration model is in a different database instance from the one in which you are developing the configuration model. For information, see Exploding BOM Models in Oracle Applications., page 5-10

- Refresh data in the CZ schema as production BOM and Inventory data changes. For information, see Refreshing Imported Data., page 5-16

- Run the concurrent programs to migrate Item and Model structure data from one schema into the CZ schema. For more information, Migrating Data., page 6-1

- Verify that after populating or refreshing the CZ schema the BOM Model data is correct by viewing the Item Master area of the Repository in Oracle Configurator Developer. For information, see the *Oracle Configurator Developer User's Guide*.

- Synchronize BOM Model data in the CZ schema with production Inventory and BOM data if the import server or publication target have changed by running concurrent programs for that purpose. For information, see Synchronizing BOM Model Data., page 7-2

- If you plan to base your configuration model on legacy data, prepare that data and custom extraction and load programs so the data can be transferred to the CZ schema. For information, see Custom Import., page 5-30

- Migrate Functional Companions that were developed prior to 11*i*10 to Configurator Extensions. For more information see Migrate Functional Companions., page C-32

- Purge tables in the CZ schema if your database gets too large and fails to perform adequately. It is recommended that you purge tables on a regular basis. The Purge Configurator Tables concurrent program deletes those records that are marked for deletion. The Purge Configurator Import Tables, Purge To Date Configurator Import Tables, and Purge To Run ID Configurator Import Tables concurrent programs delete data in the CZ_IMP tables, and the corresponding data in the CZ_XFR_RUN_INFOS, and CZ_XFR_RUN_RESULTS control tables. For more information see Purging Configurator Tables., page 8-2

- Delete old configuration data by running the DELETE_CONFIGURATIONS API. For more information, see DELETE_CONFIGURATION, page 17-51.

Optional Database Tasks

Optional tasks for providing additional flexibility in your Oracle Configurator implementation include:

- Use PL/SQL to modify nodes created in Configurator Developer and BOM Model Item descriptions to use Multiple Language Support (MLS). For more information see Multiple Language Support., page 14-1

- Write Configurator Extensions designed to populate CZ table fields with configuration data that cannot be directly inserted using runtime Oracle Configurator. For more information, see the *Oracle Configurator Extensions and Interface Object Developer's Guide*, and Migrate Functional Companions, page C-32.

- Design custom configuration attributes and attach them to certain nodes of configuration models. For more information, see the *Oracle Configurator Methodologies* documentation.

- Write legacy rules in Constraint Definition Language (CDL) format and import the rules into the CZ schema. For CDL information, see the *Oracle Configurator Constraint Definition Language Guide*. See Rule Import, page 5-21 for rule import information.

Integration Tasks

Integration tasks enable Oracle Configurator to work with a particular host application.

Required Tasks for All Integrations

These tasks must be performed for all integrations of Oracle Configurator with a host application.

- Set profile options to integrate and set behavior of Oracle Configurator within Oracle Applications. For a listing of profile options that affect Oracle Configurator, see the *Oracle Configurator Installation Guide*.

- Verify and set the Apache and JServ properties for your host application that affect the runtime Oracle Configurator. See the *Oracle Configurator Installation Guide* for more information.

- Verify and set properties of the Oracle Configurator Servlet for your host application. See the *Oracle Configurator Installation Guide*for more information.

- Test the integration of Oracle Configurator in the host application running in a Web browser.

Optional Integration Tasks

These tasks provide additional aspects of integration between Oracle Configurator and a host application, and apply to both custom and predefined integrations.

- Provide pricing and **ATP** support for the runtime Oracle Configurator by setting switches in the file `cz_init.txt`. See Pricing and ATP in Oracle Configurator., page 13-1

- Enable Multiple Language Support (MLS). For details see Multiple Language Support., page 14-1

- Set up the Model structure and Configurator Extensions for configuration attributes. See the *Oracle Configurator Methodologies* documentation.

Tasks for Custom Integration

These tasks (in addition to the required tasks listed in Required Tasks for All Integrations, page 1-4) must be performed if you are integrating Oracle Configurator with a custom host application. A custom host application is one that does not provide any predefined integration with Oracle Configurator.

- Manually install servlet, media, and HTML files and verify that these files are in the correct location. See the *Oracle Configurator Installation Guide* for more information.

- Tailor the initialization message that invokes the runtime Oracle Configurator. For details, see Session Initialization., page 9-1

- Create and install a servlet that handles the runtime Oracle Configurator's XML termination message, which contains configuration output data. For details, see Session Termination., page 10-1

- Set up a return URL for the servlet that handles the termination message, and add it

to the initialization message. For details, see Session Initialization., page 9-1

Model Development Tasks

Model development tasks enable you to extend a BOM Model by adding additional structure, rules, UIs, and publishing your configuration model to a host application.

Required Tasks for Model Development

These tasks must be performed so that you can create Models or add additional structure, rules, and UIs to BOM Models.

- Design configuration models with performance in mind. See the *Oracle Configurator Performance Guide* for guidelines.
- Verify the imported data in Configurator Developer if you are developing a configuration model based on existing data in Oracle Applications **Bills of Material** and Inventory. See Database Tasks, page 1-2 for additional tasks needed to populate the CZ schema.
- Define the structure, rules, and user interface in the Model's Workbench. See the *Oracle Configurator Developer User's Guide* for more information.
- Generate logic to create the structure and rules of the configuration model. Generating logic is also used to help debug some issues. Rerun this procedure after you have completed the following activities:
  - Changed rule definitions
  - Changed the Model structure
- Select the Refresh option on the UI Workbench page or the UI Refresh Status on the General Workbench page to update a User Interface with the latest modifications to the User Interface definitions and customizations. Rerun this procedure after you have completed the following activities:
  - Changed the Model structure
  - Refreshed your BOM-based model
- Unit test your configuration model before publishing it. See the *Oracle Configurator Developer User's Guide*.
- Create a publication for the configuration model to appropriate host applications. See the *Oracle Configurator Developer User's Guide*.

- Define the configuration model's applicability parameters in preparation for publishing the configuration model so that it can be accessed by a host application. See the *Oracle Configurator Developer User's Guide.*

- Assign each publication to one Model and the appropriate usages to control when and if the usages are invoked by the host applications. See the *Oracle Configurator Developer User's Guide* for additional information.

- Publish configuration models for availability to host applications. For information, see Publishing Configuration Models, page 16-1 and the *Oracle Configurator Developer User's Guide.*

- Republish the configuration model if the Model's structure, rules, or UI change. For more information, see the *Oracle Configurator Developer User's Guide*.

Optional Tasks for Model Development

The following tasks can be performed to provide additional Model functionality.

- Write Configurator Extensions to extend the functional capabilities of your configuration model beyond what is implemented in Oracle Configurator Developer. For information on writing Configurator Extensions see the *Oracle Configurator Extensions and Interface Object Developer's Guide* , *Oracle Configurator Developer User's Guide* and the *Oracle Configurator Methodologies* documentation.

- Change the default behavior of locking Models or UI Content Templates. For more information, see the *Oracle Configurator Developer User's Guide*.

- Set the Effectivity Date Filter if you want to filter ineffective Model structure nodes and rules when working in Configurator Developer . For more information about the Effectivity Date Filter, see the *Oracle Configurator Developer User's Guide*.

- Run the Add Model Names to Configurations by Model Items, page C-35 or Add Model Names to Configurations by Model Product Key, page C-37 concurrent program in order to restore configurations, made prior to your upgrade, against updated migrated Models.

Deployment Tasks

Deployment involves making a runtime Oracle Configurator available to end users. The following tasks complete the deployment of a runtime Oracle Configurator either embedded in a host Oracle Application or in a custom host application.

Required Tasks for All Deployments

The following tasks are required for the runtime Oracle Configurator to use the

currently supported user interfaces. (As of this release, DHTML UIs are no longer supported.)

- Turn off pop-up blockers.
- Recommended screen resolution is 800 X 600 or greater. This depends on how you have generated the Components Tree user interface in Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for details.
- System test the configuration model by accessing it from the host application.
- Optimize the performance of the production environment by:
  - Adjusting system size or setting up the database and application tiers on multiple server computers.
  - Tuning components of the Oracle Configurator architecture on the client system, such as browser settings, swap space, and memory
  - Adjusting Web server configuration settings
  - Determining whether you should load balance the Apache Web listener
  - Determining whether you should load balance across CPUs on a multi-CPU machine

  For details see the *Oracle Configurator Performance Guide*.
- Run LoadRunner to determine response time, CPU utilization, number of transactions per hour, throughput and hits per second. See the *Oracle Configurator Performance Guide* for load testing.

Optional Tasks for Deployment

These tasks can be performed to maximize performance, usability, and functionality when your configuration model is deployed to end users.

- Consider preloading a configuration model for improved performance. For details see the *Oracle Configurator Performance Guide*.
- Load balance the Apache Web listener (HTTP). For details see the *Oracle Configurator Performance Guide*.
- Set up Secured Sockets Layer (SSL) if you want to create a secure connection between a client and server system. For details see Load Balancing and Secure Sockets Layer., page 20-6 For additional SSL information, seel the Oracle Support Web site

- Set up a dedicated Jserv for running Oracle Configurator, if you want to take advantage of Oracle Applications Java Caching Framework (OAJCF). For more information about Oracle Applications Java Caching Framework, see the *Oracle Configurator Performance Guide*.

- Adjust the `ApJServVMTimeout` setting that affects the amount of time to wait for the JVM to start up and respond. See the *Oracle Configurator Installation Guide* for details.

- Pricing behavior must be set for Item price display type and price data update method. For more information, see Controlling Pricing and ATP in a Runtime Oracle Configurator., page 13-12

- Consider setting up firewalls, using routers, and separate computers to protect unauthorized access to your servers. For more information, see Deployment Considerations., page 20-1

Tasks for Custom Deployments

If you are implementing a custom deployment, then consider the following:

- Create a UI that adheres to the Oracle guidelines. See *Oracle Configurator Developer User's Guide* for User Interface information.

- Create online Help for the runtime Oracle Configurator. See *Oracle Configurator Developer User's Guide* for generic runtime information.

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The table below lists other conventions that are also used in this guide.

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.

Convention	Meaning
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a new term, a term defined in the glossary, specific keys, and labels of user interface objects. Boldface type also indicates a menu, command, or option, especially within procedures
italics	Italic type in text, tables, or code examples indicates user-supplied text. Replace these placeholders with a specific value or string.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
>	The left bracket alone represents the MS DOS prompt.
$	The dollar sign represents the DIGITAL Command Language prompt in Windows and the Bourne shell prompt in Digital UNIX.
%	The per cent sign alone represents the UNIX prompt.
`name()`	In text other than code examples, the names of programming language methods and functions are shown with trailing parentheses. The parentheses are always shown as empty. For the actual argument or parameter list, see the reference documentation. This convention is *not* used in code examples.
&	Indicates a character string (identifier) that can display text dynamically in Configurator Developer or a runtime Oracle Configurator. For example, "&PROPERTY" can be used to dynamically construct and display a Property of a Model structure node.

Product Support

The mission of the Oracle Support Services organization is to help you resolve any issues or questions that you have regarding Oracle Configurator Developer and Oracle Configurator. Navigate to the Knowledge area of My Oracle Support Browse to the Knowledge subtab> Oracle E-Business Suite > Order Management> Configurator > All of Configurator.

You can also find product-specific documentation and other useful information using Oracle Applications Documentation, on the Oracle Technology Network.

For a complete listing of available Oracle Support Services and phone numbers, see the Oracle Support Web site.

Troubleshooting

Oracle Configurator Developer and Oracle Configurator use the standard Oracle Applications methods of logging to analyze and debug both development and runtime issues. These methods include setting various profile options and Java system properties to enable logging and specify the desired level of detail you want to record.

For more information about logging, see:

- The *Oracle E-Business Suite System Administrator's Guide* for descriptions of the Oracle Applications Manager UI screens that allow System Administrators to set up logging profiles, review Java system properties, search for log messages, and so on.
- The *Oracle E-Business Suite Developer's Guide*, which includes logging guidelines for both System Administrators and developers, and related topics.
- The *Oracle Application Framework Developer's Guide*, which describes the logging options that are available via the Diagnostics global link. This document is available in the Applications Documentation, on the Oracle Technology Network.

2

Configurator Architecture

This chapter describes the elements of the Oracle Configurator product and how they fit together.

This chapter covers the following topics:

- Overview
- Introduction
- Runtime Oracle Configurator
- Oracle CZ Schema
- Oracle Configurator Developer
- Multi-Tier Architecture

Overview

This chapter presents the elements of the Oracle Configurator product and how they fit together, including information about:

- Runtime Oracle Configurator, page 2-3
- Oracle CZ Schema, page 2-7
- Oracle Configurator Developer, page 2-8
- Multi-Tier Architecture, page 2-9

Introduction

Oracle Configurator Developer is both a development and maintenance environment used to create, modify, and unit test configuration models and custom Oracle runtime configurator pages. The runtime Oracle Configurator, Oracle Configurator Developer, and CZ schema run as part of the Oracle Applications *e*Business Suite in a multi-tier

architecture.

Oracle Configurator Developer is a thin client development environment that connects directly to the CZ schema.

Both the runtime Oracle Configurator and Oracle Configurator Developer run in a browser. The Oracle Configurator (the application itself) runs on the application server machine with the internet application server brokering the processes and http connection.

The runtime Oracle Configurator and Oracle Configurator Developer:

- Are HTML based
- Operate within the Oracle Applications (OA) Framework
- Are Self Service Web applications

Oracle Configurator consists of the following elements:

- Oracle Configurator Developer
- CZ schema within the Oracle Applications database
- Runtime Oracle Configurator

Oracle Configurator Developer includes the following OA Framework features:

- Based on J2EE standards
- Facilitates access by the disabled community
- Multiple Language Support (MLS)
- Multi-currency support
- Reusable UI components

Additionally, Oracle Configurator Developer leverages the latest Oracle Application Server technology, such as:

- Caching
- Event Handling
- Security
- State Management
- XML Based Declarative UIs

    - Optimized HTML UI rendering
    - Presentation is separate from business logic
- Business Components for Java (BC4J)
    - Business logic encapsulation
    - Optimized DB interaction
    - Scalability and performance
- Message-service EJB Architecture
    - Full support for transactions, fail-over and multi-tier deployment
    - Minimizes inter-tier traffic

The runtime Oracle Configurator, Configurator Developer, and the CZ schema are installed with Oracle Applications Release 12 by running Oracle Rapid Install.

Runtime Oracle Configurator

The runtime Oracle Configurator enables end users to select options interactively in a Web browser.

It is also possible to run Oracle Configurator as a programmatic background process, such as when an end user changes the quantity of a configured item. The background process validates the configuration without requiring further end-user interaction.

Access

End users access the runtime Oracle Configurator by logging into an application that hosts Oracle Configurator. When the user requests that the host application configure something, the host application invokes Oracle Configurator, which then becomes the foreground application during a configuration session. At the end of a configuration session, the Oracle Application dialog page is displayed before the host application returns to the foreground.

There are several factors that affect the way that you can enable users to access the runtime Oracle Configurator:

- Type of Host Application, page 2-4
- Login to Host Application, page 2-4
- Invocation of Oracle Configurator by Host Application, page 2-4

- Incorporation of Oracle Configurator in the Host Application's UI, page 2-5

These factors are described in the following sections.

Type of Host Application

The host application for the runtime Oracle Configurator can be one of the following:

- An application that is part of Oracle Applications, which you reach through the E-Business Suite home page. Examples are: Oracle Order Management, *i*Store, and Oracle Contracts. Oracle Configurator Developer is also a host application. .

- A custom application that provides its own user interface, and at runtime communicates with the Oracle Configurator engine through the Configuration Interface Object (CIO).

Login to Host Application

End users of the host application can log in by one of the following methods:

- If the host application is part of Oracle Applications, then users log in to the E-Business Suite home page with a user ID and password that are authenticated by Oracle Applications. This process generates an ICX session ticket, which contains the session authentication information that is used by the runtime Oracle Configurator.

- If the host application is not part of Oracle Applications, then after a user logs in to the host application, that application must specify user ID, password, and database identification when it invokes the runtime Oracle Configurator.

Invocation of Oracle Configurator by Host Application

All host applications send an initialization message to start the runtime Oracle Configurator, and specify parameters of the message to control the initial state of the runtime Oracle Configurator. Oracle Configurator processes the initialization message in the following way:

1. The host application sends the initialization message, which is in XML, to the URL of the Oracle Configurator Servlet. The host application obtains this URL from the profile option BOM: Configurator URL of UI Manager. See the *Oracle Configurator Installation Guide* for details about setting profile options. The Oracle Configurator Servlet is described in Oracle Configurator Servlet , page 2-6.

   Oracle Configurator can be invoked programmatically by the host application, without user interaction. This is called batch validation, which is described in Batch Validation, page 11-1.

   1. If the initialization message is wrapped in the `<batch_validate>` element,

then the Oracle Configurator Servlet runs Oracle Configurator in a batch validation session.

2. If the initialization message is not intended for batch validation, then Oracle Configurator determines which type of user interface to render, based on the value of the initialization parameter ui_type, page 9-36.

   The user interface for the runtime Oracle Configurator can use one of the styles described in Runtime UI Types, page 2-6. It can also use a completely custom UI, if the host application provides its own user interface, and its own code to communicate with the Oracle Configurator engine directly, through the Oracle Configuration Interface Object (CIO).

2. Oracle Configurator processes the parameters in the initialization message, and begins a configuration session, rendering the specified runtime Oracle Configurator. The parameters determine the initial state of the configuration session, specifying which model to configure and a variety of other configuration data. The particular selection of parameters and values depends on the requirements of the host application. See Session Initialization, page 9-1 for details.

Incorporation of Oracle Configurator in the Host Application's UI

Invocation results in the host application incorporating the user interface for the runtime Oracle Configurator into its own user interface in one of the following ways:

- Standalone page: Oracle Configurator occupies all of a standalone page, in a page separate from that used by the host application. Examples: Oracle Order Management and Oracle Configurator Developer.

- Frame: Oracle Configurator occupies a frame that is embedded in the page used by the host application. Example: Oracle *i*Store.

- Region: Oracle Configurator occupies a region that is embedded in a page used by the host application. Only possible if the host application is a member of Oracle Applications that is constructed with the Oracle Applications Framework. For more information about the Oracle Applications Framework, see the Oracle Application Framework Documentation Resources, Release 12, available in the Oracle Applications Documentation, on the Oracle Technology Network.

- Custom container: Oracle Configurator occupies a JavaServer Page that you specify when you publish your Model.

Oracle Configurator Security on Publicly Accessible Web Servers

For information and recommendations on preparing the deployment of Oracle Configurator on publicly accessible Web servers, see Deployment Considerations, page

20-1.

Runtime UI Types

Depending on your runtime UI requirements, you can deploy the following types of runtime Oracle Configurators:

- User Interfaces that are based on the OA Framework, deployed as part of the E-Business Suite, and launched from other Oracle Applications. For a list of Oracle Applications that integrate with Oracle Configurator, contact your Oracle representative'. For details about creating generated UIs, see the *Oracle Configurator Developer User's Guide*.

- Legacy Configurator User Interfaces (Java **applet**) from previous releases of Oracle Configurator. These legacy UIs cannot be edited using the HTML-based Oracle Configurator Developer. For details, see the Oracle Configurator documentation from previous releases and the *Oracle Configurator Installation Guide*.

- The Generic Configurator User Interface.

Oracle Configurator Servlet

The Oracle Configurator Servlet contains the machinery used to support:

- Batch validation

- Legacy Configurator user interfaces

> **Note:** The inclusion of the Oracle Configurator Servlet in this release provides compatibility for host applications that were already integrated with Oracle Configurator before the adoption of the Oracle Applications Framework. See Invocation of Oracle Configurator by Host Application, page 2-4 for an example of this integration. All other areas of Oracle Configurator provide integration through the Oracle Applications Framework, as described elsewhere in this chapter. For more information on the Oracle Applications Framework, see the Oracle Application Framework Documentation Resources, Release 12, in the Oracle Applications Documentation, on the Technology Network.

The Oracle Configurator Servlet is responsible for rendering legacy Configurator user interfaces and brokering communication between the configuration model, the database, and the client browser.

The OC Servlet consists of the following elements:

- UI Server, page 2-7

- Configuration Interface Object (CIO), page 2-7
- Oracle Configurator Engine, page 2-7

The OC Servlet runs on Oracle Application Server, which includes the Apache Web Server. The behavior of the OC Servlet can be customized by setting servlet properties. The properties of the OC Servlet are described in the *Oracle Configurator Installation Guide*. Information about setting servlet properties is presented in the *Oracle Configurator Performance Guide*.

UI Server

The UI Server is an element of the OC Servlet that is not used by Oracle Configurator when it renders a user interface in the Oracle Applications Framework.

The UI Server that processes user input from a client user interface and renders back the UI for display to the end user based on information received from the Oracle Configurator engine. The UI Server provides a common level of support for user interfaces (Java applet) that are not created by the HTML-based Oracle Configurator Developer.

Configuration Interface Object (CIO)

The CIO is an **API** layer that handles communication between the Oracle Configurator engine and the UI. The API methods of the CIO can be used to access the configuration model and Oracle Configurator behaviors. Configurator Extensions and custom UIs communicate with the Oracle Configurator engine through the CIO.

For more information see the *Oracle Configurator Extensions and Interface Object Developer's Guide*.

Oracle Configurator Engine

The Oracle Configurator engine validates user selections and provides results based on the compiled structure and rules of a configuration model.

The Oracle Configurator engine has no public API and cannot be modified.

Oracle CZ Schema

The CZ schema consists of Configurator (CZ) tables in the Oracle Applications Release 12 database that are accessed by both the runtime Oracle Configurator and Oracle Configurator Developer.

The CZ schema is organized into subschemas that store:

- Imported data from other Oracle Applications database tables
- Settings that control the behavior of Configurator processes

- Data that defines the Model structure, rules, and UI of configuration models
- Saved configurations

Oracle Configurator Developer stores the complete definition of the User Interface in the CZ schema, where it is available to both Oracle Configurator Developer and a runtime Oracle Configurator.

See CZ Subschemas, page D-1 for a listing of the tables that are in each of the subschemas. For more information about the CZ schema data model, see the Oracle Integration Repository.

Oracle Configurator Developer

Oracle Configurator Developer:

- Allows creating, organizing, managing, and publishing Models
- Includes tools for generating runtime Configurator User Interfaces
- Allows users to define configuration rules

Access

Users access Configurator Developer by logging into Oracle Applications and selecting the appropriate responsibility. The following responsibilities are predefined and available with initial installation:

- Oracle Configurator Developer
- Oracle Configurator Administrator
- Oracle Configurator Viewer

For more information on accessing Configurator Developer, see Controlling the Development Environment, page 15-1.

Types of Configuration Models

Users of Configurator Developer can create a configuration model using only the structural elements (Model, Components, Features, Options) available in Configurator Developer. This is called a Developer Model and might be used to create a standalone or prototype configuration.

If the configuration model is based on an imported **ATO** or **PTOBOM** Model, then users of Configurator Developer can extend the imported Model with Configurator Developer structure to create guided buying or selling questions, and additional internal structure to support rule definition.

Users of Configurator Developer can also extend the behavior of configuration models beyond what can be implemented in Oracle Configurator Developer by creating Configurator Extensions. Configurator Extensions are built with custom or provided Java code that uses the fully supported, fully documented Java API methods of the CIO. Implementers create Configurator Extensions and then connect them to configuration models in Configurator Developer.

Unit Testing

To unit test a configuration model, you can access the runtime Configurator UI as a test environment directly from Configurator Developer to create configurations. You can also use the Model Debugger in Configurator Developer to unit test new configurations or restore saved configurations. Testing uses the same application architecture as a deployed runtime Configurator.

When unit testing, you can:

- Specify testing session parameters, such as Effectivity dates and a Usage
- Save and restore configurations
- Run Configurator Extensions
- Display pricing and **ATP** information

Testing from Configurator Developer through Oracle Applications does not involve running the host application where your configuration models are deployed, such as Order Management. For more testing information, see the *Oracle Configurator Developer User's Guide*.

Multi-Tier Architecture

Oracle Applications architecture is a framework for multitiered, distributed computing. Oracle Application Framework fits into a three-tier architecture. The three tiers are:

- Application
- Client
- Database

Oracle Application Framework also fits into a four-tier architecture.

The four tiers are:

- Application
- Client

- Database
- Web

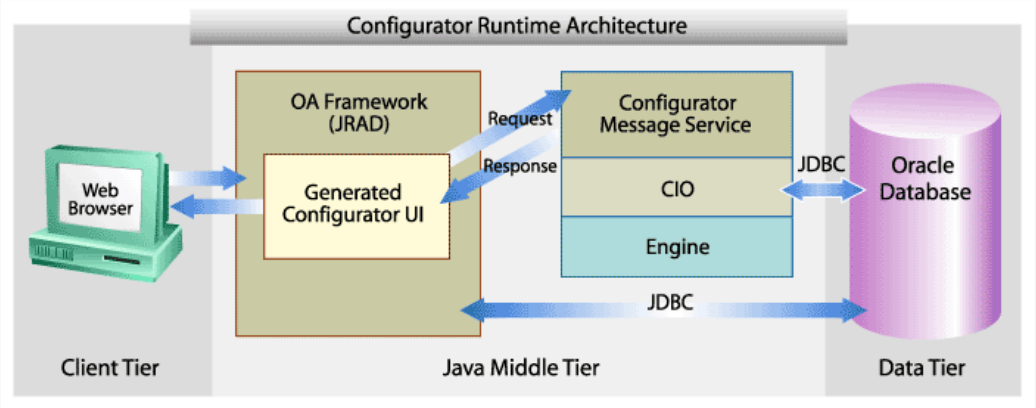
For more information about the Oracle Application Architecture, see the *Oracle E-Business Suite Concepts* documentation and the Oracle Application Framework Documentation Resources, Release 12, in Oracle Applications Documentation, on the Oracle Technology Network.

Runtime Oracle Configurator

The elements of a runtime Oracle Configurator that span the four tiers are shown in Four tier Architectural Overview of a Runtime Oracle Configurator, page 2-10.

The following table shows the two way communication between the Client tier and the Web tier. The Web tier contains custom Java Server Pages and the OA Framework that contains the Generated Configurator UIs. The Web tier sends requests to the Application tier that consists of the Configurator Message Service, the CIO, and the Engine. The Application tier then sends responses back to the Web tier. There is two way communication between the Application tier and the Data tier. The Data tier is the Oracle Database.

Four tier Architectural Overview of a Runtime Oracle Configurator



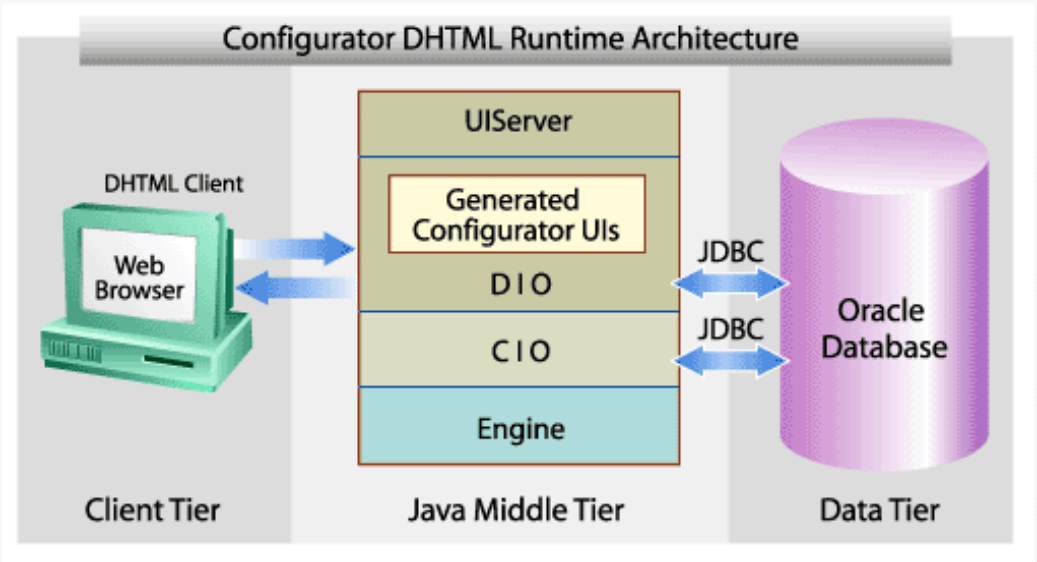
During an interactive runtime session, the Web tier contains the displayed UI. The Configurator Messaging service in the Applications tier uses Enterprise Java Beans to handle requests from the displayed page on the Web tier.

The elements of a runtime Oracle Configurator that span the three tiers are shown in Three tier Architectural Overview of a Runtime Oracle Configurator, page 2-11.

Three tier Architectural Overview of a Runtime Oracle Configurator , page 2-11 illustrates the two way communication between the Client tier and the Java Middle tier. The Java Middle tier is made up of the UI Server, the Generated Configurator UIs, the DIO, the CIO and the Engine. It also illustrates the two way communication between

the Java Middle tier and the Data tier via JDBC. The Data tier is the Oracle database

Three tier Architectural Overview of a Runtime Oracle Configurator

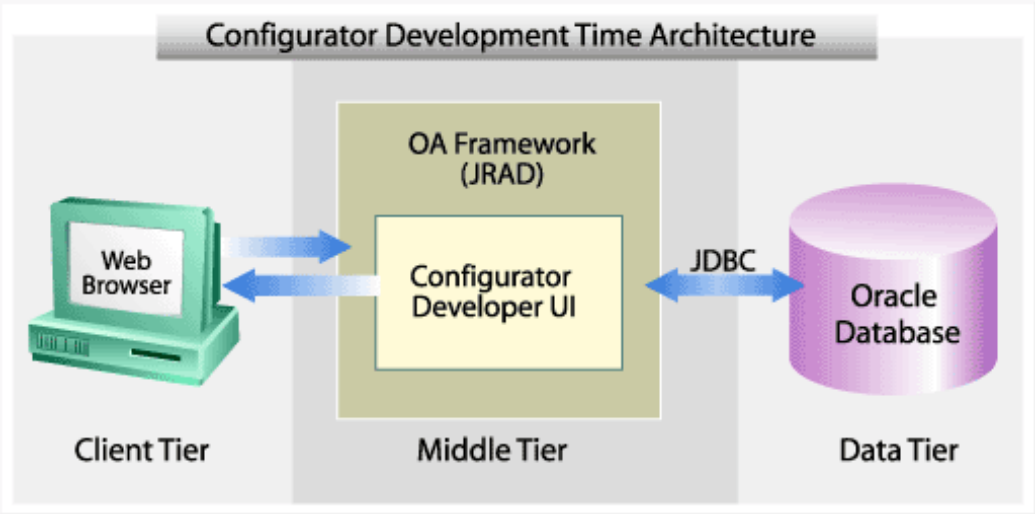


Oracle Configurator Developer Three Tiers

During development, Configurator Developer runs on a three-tier architecture, with the thick web tier accessing the database as shown in Three tier Architectural Overview of Oracle Configurator Developer, page 2-12.

Three tier Architectural Overview of Oracle Configurator Developer, page 2-12 illustrates the two way communication between the Client tier and the Middle tier. The Client tier is the Web browser. The Middle tier consists of the OA Framework, and the Configurator Developer UI. There is two way communication between the Middle tier and the Data tier via JDBC. The Data tier is the Oracle database.

Three tier Architectural Overview of Oracle Configurator Developer



Configurator Developer is a thin-client development environment that connects directly to the CZ schema. Configurator Developer is built on the Oracle Applications Framework and leverages the latest Oracle Application Server technology that allows for XML Based Declarative UIs, Business Components for Java (BC4J), and Message-Service EJB architecture.

3

Database Instances

This chapter describes the uses to which databases are put when implementing Oracle Configurator, and specifics about using multiple database instances.

This chapter covers the following topics:

- Overview
- Database Uses
- Multiple Database Instances
- Model Development
- Maintenance
- Production

Overview

Whether your implementation project uses a single or two separate Oracle Applications database instances, the database serves multiple purposes during an Oracle Configurator implementation. The topics in this chapter include:

For details about the CZ schema within an Oracle Applications database instance, see Configurator Architecture, page 2-1 and The CZ Schema, page 4-1.

Database Uses

During an Oracle Configurator implementation, the Oracle Applications database is used for:

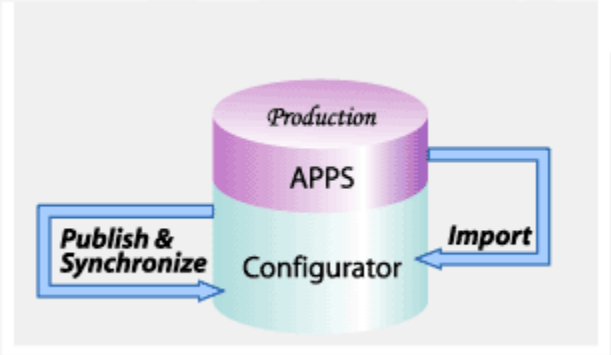
- Migrating or importing data into the CZ schema
- Running Oracle Configurator Developer to create configuration models
- Unit and system testing configuration models
- Publishing configuration models
- Running a production Oracle Configurator
- Storing Items, BOM Models, and saved configurations

During an Oracle Configurator implementation and deployment, Oracle supports using either a single database instance for all operations, or multiple development instances and one production instance.

- Development, page 3-7 instances can serve as:
  - Import target
  - Publication source and target
  - CZ schema migration source and target
  - Model migration source and target
  - BOM Model synchronization source or target
- Production, page 3-8 instances can serve as:
  - Import source
  - Publication target
  - Migration source
  - BOM Model synchronization source

The Single Database Environment, page 3-3 illustration shows that a single database environment can be used to import Oracle Application data into the CZ schema , as well as publish and synchronize data.

Single Database Environment



A publication's details and applicability parameters determine a configuration model's unique deployment. For more information on deploying a configuration model, see Publishing Configuration Models, page 16-1.

To support Oracle Configurator implementations on separate development and production database instances, Oracle provides the means for moving and synchronizing data across the instances. For more information about moving data, see Populating the CZ Schema, page 5-1 and Migrating Data, page 6-1. For more information about synchronizing data, see Synchronizing Data, page 7-1.

For more information about implementing Oracle Configurator in two separate database instances, see Multiple Database Instances, page 3-3.

Multiple Database Instances

Once a configuration model is deployed, separate database instances can ensure that maintenance or instabilities in the operations of the development database instance do not interfere with end-user access or ongoing maintenance of the application that is in production use.

> **Note:** Publishing Models from *more* than one development instance to the same production instance can cause unresolvable problems with data synchronization.

Although the following operations can be accomplished on a single database instance, they commonly involve separate development and production database instances:

- Importing or migrating data from a production database instance into the development CZ schema
- Publishing configuration models from a development instance to a production CZ schema

- System testing configuration models in a production database instance

When working with multiple database instances, the instances in which the user runs Oracle Configurator Developer to create and develop models, are the *local* or source database instances. The database instance to which Models are published and used in production is the *remote* or target database instance.

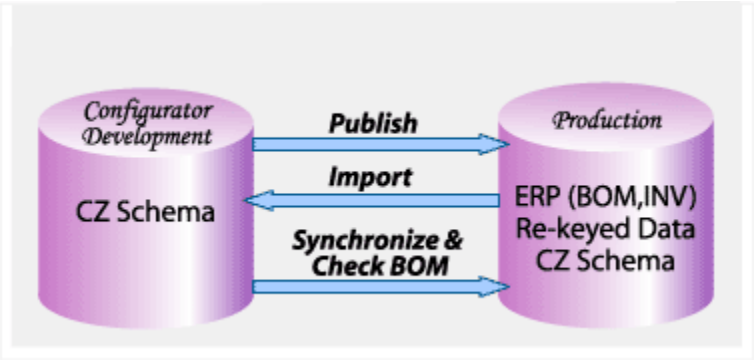
Model Availability on Multiple Database Instances

Although it is possible to implement and deploy Oracle Configurator using only one database instance, many projects use multiple database instances to distinguish between Model development and production use. Models can be copied into multiple development instances. Copying Models from any instance to a development instance is known as migrating Models. Models are published to a production instance, but should not be migrated to a production instance.

Once a Model is migrated, its structure, rules and User Interfaces can be uniquely expanded in each development instance. A single Model from a designated instance is then published to the production environment. See Migrating Models, page 6-3 for more information.

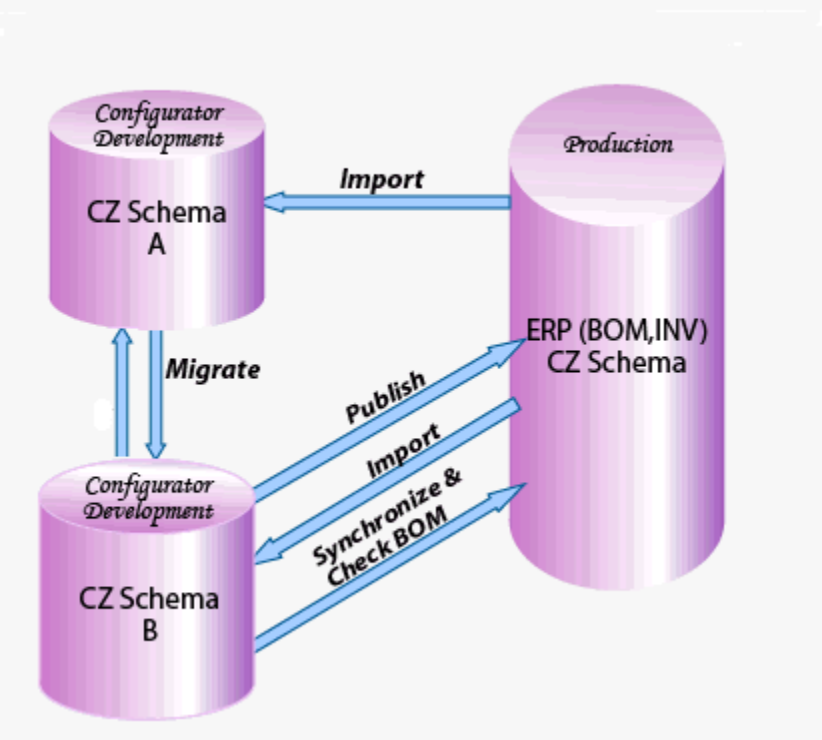
The illustrationTwo Database Environments, page 3-4 shows that a production database is used to import Oracle Application data into a development database. After creating rules, UIs, and extending the Model structure, the Model is published to the production database. Synchronization is done from the development database with the production database.

Two Database Environments



The Multiple Database Instances illustration shows Oracle Application data can be imported from a production database to development databases. It also shows that Models can be migrated from one development database to another. After creating rules, UIs, and extending the Model structure, the Model is published to the production database. Synchronization is done from the development database with the production database.

Multiple Database Instances



See Synchronizing Migrated Model Data, page 6-8 for information on synchronizing a migrated Model's data.

See Migrating Models, page 6-3 for scenarios using separate development and production database instances.

Import Source and Target

To develop a BOM-based configuration model, BOM Model data must be imported into the CZ schema. The imported data used to develop a runtime Oracle Configurator should be production data. The production database serves as the import source. The development instance serves as the import target. For information about data import, see Populating the CZ Schema, page 5-1.

Publication Source and Target

Configuration models must be published from the development database instance to be available for system testing or production use in the same or a different database. You can delete publications on the target instance from Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for additional publishing information. For information about publishing, see Publishing a Configuration Model, page 16-10.

Oracle strongly recommends that all source and target instances which participate in publishing be located on the same local area network. When publishing over a wide

area network, performance can be degraded by network factors.

If you change the publication source or target (by running the Modify Server Definition, page C-14 concurrent program) or use a cloned source or target, then you must synchronize the publication data. See Synchronizing Data, page 7-1. If the BOM Model data changes in Oracle Bills of Material, or you modify the Model structure or UI in Configurator Developer, then you must republish the Model.

A previously defined remote publishing target instance can be converted to a development instance. See Converting a Remote Target to a Development Instance., page 3-9 For information about what happens to existing publications when a remote target is converted to a development instance, see Source and Remote Publications., page 16-5

Decommissioning a Database Instance

Decommissioning a production database instance (target) causes synchronization problems. For more information on synchronization, see Synchronizing Data, page 7-1 .

Migration Source and Target

When you need to move your Configurator implementation or deployment from one database instance to another, you may need to migrate configuration model data. The instance where the Model is migrated from is referred to as the source instance. The instance where the Model is migrated to is referred to as the target instance. For more information about migration, see Migrating Data, page 6-1.

> **Note:** The installed languages must be the same on both the source and target instances.

For migration source and target patch level information, see the current release or patch information for Oracle Configurator in Oracle Applications Documentation, on the Oracle Technology Network.

BOM Synchronization Source and Target

In cases where the import source or publication target change, it may be necessary to synchronize the BOM-based configuration model with the corresponding production BOM. For more information about BOM synchronization, see Synchronizing Data, page 7-1.

Linking Multiple Database Instances

When creating an empty database or repurposing an existing one to serve as a source or target of data operations across two database instances, the databases must be linked. Defining and enabling the remote server sets up the necessary database links between the source and target databases.

See Server Administration, page B-3 for general information on setting up database links. For details on running the concurrent programs, see Define Remote Server, page C-10, and Enable Remote Server, page C-12.

Instance and Host System Names

Multiple database instances can exist on a single or separate host systems. Both the database instance and the host system have a name. The name of the database instance and host system are relevant in all the uses listed in Reasons for Multiple Database Instances, page 3-4.

In this book, the database instance you are connected to or logged into is the local or current database instance, and the local system is the local host. Other instances, whether on the local host system or a different remote system, are remote instances in relation to the local instance.

The local database instances can serve as:

- Target database for data migration
- Target database for data import
- Source database for publishing configuration models
- Original database for creating a clone

Remote database instances can serve as:

- Source database for data migration
- Source database for data import
- Target database for publishing configuration models

The SID is used to identify the database instance that Oracle Configurator Developer uses. The database instance name is also known as the local name. The database instance and host names are required in various places for the correct operation of Oracle Configurator Developer, the CZ schema, and Configurator concurrent programs. The SID is specified during Rapid Install. For more information, see the *Oracle E-Business Suite Installation Guide: Using Rapid Install* guide.

Model Development

A development database instance is one in which you create your configuration model using Configurator Developer.

> **Note:** There may be multiple development database instances. Configuration models that are available to end users should only be

published from a single development environment. Publishing Models from multiple development instances to a single test or production instance could result in:

- Publications with overlapping applicability parameters
- Multiple development environments leading to confusing publication history. Publication history is maintained on the development environment.
- Overwriting a configuration model's snapshot of its Item Master. When a configuration model is published, the publication has a snapshot of the development environment's Item Master. If a configuration model is published from a different development environment, then the snapshot of its Item Master overwrites the original Item Master.

Unit testing is initiated from Configurator Developer by launching either the Model Debugger or a generated User Interface. Unit testing enables the implementer to test configuration rules and UI functionality in the development database instance. Unit testing ensures that rules and UI modifications work as desired. For additional information, see the *Oracle Configurator Developer User's Guide*.

When you upgrade the release version of Oracle Configurator that your runtime Oracle Configurator runs against, you start by upgrading the CZ schema. For information about updating your CZ schema, see the *Oracle Configurator Installation Guide*.

Maintenance

Oracle Configurator data is maintained in the maintenance environment. A maintenance environment is similar to a development environment because it requires many of the same operations such as upgrading the CZ schema, refreshing configuration data, fixing and improving configuration models, and periodically republishing the models. It is important to synchronize any changes in the maintenance database instance with the development database instance for the next release of your runtime Oracle Configurator. For more information on synchronization, see Synchronizing Data, page 7-1.

Production

A production environment is one in which runtime Oracle Configurator end users use the software in a production mode. The production environment is also used for system testing.

System Testing

The system testing environment is generally the production environment and used to verify that data transfers and modifications in a deployed scenario work as desired. For example, changes to the Model structure in Oracle Configurator Developer should propagate to the host application such as Order Management.

System testing includes publishing the configuration model and UI, accessing it using at least one host application, and specifying various effective dates. System testing tests:

- Performance of the configuration model
- End-user access
- Security
- Integration customizations

Deploying a Model

To prepare for deploying the configuration model to your production environment, you must consider integration with other applications, perform unit testing, and system testing. For additional information see the *Oracle Configurator Developer User's Guide*.

If the development database and the production database are not on the same machine, then the production database server must be defined and enabled. For more information on defining a remote server, see Define Remote Server, page C-10.

Before you publish the configuration model, purging records flagged for deletion results in a more efficient use of computer resources. For more information about purging records, see Purging Configurator Tables, page 8-2.

For information about publishing a configuration model to a production CZ schema, see Publishing Configuration Models, page 16-1.

Converting a Publication Target Instance to a Development Instance

A publication target instance can be converted to a development instance. Once a remote target instance is converted to a development instance, it can no longer be specified as a remote instance for publishing. The converted instance can be used to publish locally. For more information, see the Convert Publication Target Instance to Development Instance, page C-8 concurrent program.

> **Note:** Converting a publication instance to a development instance does not change any existing published data.

Part 2

Data

Part 2 presents information about working with the CZ schema as described in Database Tasks, page 1-2.

4

The CZ Schema

This chapter describes the basic characteristics of the CZ schema, the schema settings and how they are used, and provides some schema maintenance tips.

This chapter covers the following topics:

- Overview
- Characteristics of the Oracle CZ Schema
- Import Tables
- Control Tables
- CZ_DB_SETTINGS Table

Overview

This chapter describes the basic characteristics of the CZ schema, the schema settings and how they are used, and provides some schema maintenance tips:

Characteristics of the Oracle CZ Schema

For a description of the CZ schema, see Oracle CZ Schema, page 2-7.

Online Tables and Integration Tables

The CZ schema contains online and integration tables. The online and integration tables

are organized into subschemas for storing the data of configuration models and saved configurations.

The online tables contain the data that is used by Oracle Configurator Developer and the runtime Oracle Configurator. Every online table that receives imported data has a corresponding import table. For example, CZ_ITEM_TYPES is populated with data from the CZ_IMP_ITEM_TYPE table during the import process. See CZ Subschemas, page 4-2 for more information about the CZ subschemas.

The integration tables consist of import and control tables. See Import Tables, page 4-3 for information about the import tables and Control Tables, page 4-9 for information about control tables. See Populating the CZ Schema, page 5-1 for information about using the integration tables.

CZ Subschemas

Both the online and integration tables of the CZ schema are organized into subschemas:

- ADMN - Administrative
- CNFG - Saved Configurations
- ITEM - Item Master
- LCE - Logic for Configuration (Generate Logic)
- PB - Publication
- PROJ - Project Structure
- RP - Repository
- RULE - Rule
- TXT - Text
- TYP - Data Typing
- UI - User Interface
- XFR - Transfer specifications and control

Additionally, there are some key table views:

- CZ_CONFIG_DETAILS_V stores selected **BOM** Model node records.
- CZ_CONFIG_ITEMS_V stores all selected node records for both BOM Models and Oracle Developer Models

See CZ Subschemas, page D-1 for a listing of tables in each subschema. For table

details, see the Oracle Integration Repository.

Public Synonyms

The CZ schema does not use public synonyms.

Schema Customization

Customizing the data model of the CZ schema is not recommended, because such customizations may not be preserved during an upgrade or migration.

Various user expansion fields in the CZ schema, such as USERNUM*n* and USERSTR*n* in the CZ_PS_NODES table, are available for custom use. The data in the user expansion fields is preserved during a schema upgrade or migration. For more information, see the Oracle Integration Repository.

Import Tables

Every import table corresponds to an online table both structurally and relationally. Each import table contains the same fields as the corresponding online table, as well as additional fields to manage the import and correlate the data with the existing data in the online table.

Import tables consist of:

- Import Control Fields, page 4-4
- Online Data Fields, page 4-7
- Surrogate Key Fields, page 4-7

Because import tables are meant to capture as much data as possible, all fields are nullable and there are no integrity constraints such as primary-key definitions, unique indexes, or foreign-key references. The import tables allow batch population of the CZ schema's online tables.

Each import table's name is similar to its online counterpart. Import tables have CZ_IMP prefix instead of just CZ_. For example, the imported data in CZ_IMP_PROPERTY populates CZ_PROPERTIES, and CZ_IMP_ITEM_TYPE populates CZ_ITEM_TYPES.

The import tables temporarily store extracted or legacy data that concurrent programs access when creating, updating, or deleting records in the CZ schema. The CZ_IMP tables are populated by running the Populate or Refresh Configuration Models concurrent programs. For more information see Populate and Refresh Configuration Models Concurrent Programs, page C-18.

For more information about:

- How data moves from sources outside the CZ schema through the import tables to the online tables, see Populating the CZ Schema, page 5-1
- Dependencies among import tables and import table codes, see Dependencies Among Import Tables, page 4-7.

Import Control Fields

Import control fields contain data that is used to manage the import process for each record. Import control data is not transferred to the online tables and is not used to resolve key values or anything else. Import Control Fields, page 4-4 describes the import control fields.

The following table shows the Import Control Fields including field name, data type, and description.

Import Control Fields

Field Name	Type	Description
RUN_ID	INTEGER	Input field that associates a record with an import run.
REC_NBR	INTEGER	Input field that is a one-up sequence number uniquely identifying each record within a RUN_ID.

Field Name	Type	Description
DISPOSITION	CHAR(1)	Output field that indicates whether the record was inserted, modified, unchanged, or rejected after an import: I = Insert M = Modify N = No change R = Rejected Null indicates that the record's disposition has not been determined. Importing rule data sets DISPOSITION in CZ_IMP_RULES and CZ_IMP_LOCALIZED_TEXTS. The success or failure of rule processing stages sets the DISPOSITION field accordingly: P = Passed R = Rejected During the key resolution stage of rule import (REC_STATUS=KRS), DISPOSITION can be: I = Rule is new in the database instance. M = Rule has previously been imported. For additional rule import information, see Rule Import, page 5-21.

Field Name	Type	Description
REC_STATUS	VARCHAR(4)	Output field that indicates the record's validation status: DUPL indicates the record is a duplicate. ERR indicates the record has not been modified or inserted into the target database table because of an error in the transfer stage. F*nnn* indicates the *nnn* field is an invalid foreign-key reference. N*nnn* indicates the required *nnn* field has null data. NULL indicates the record status is open. Once this status is set, further processing of the record is suppressed. OK indicates the data in the record now exists in the online database table. PASS indicates the record is marked for either modification or insertion after the key resolution stage. Importing rule data sets REC_STATUS in CZ_IMP_RULES and CZ_IMP_LOCALIZED_TEXTS. The rule processing stage is tracked in REC_STATUS. The following are the stages of processing rule data: CND indicates the first stage of processing rule data. This stage verifies that all required columns are populated and assigns default values for other columns. See Rule Validation, page 5-29 for a list of the required columns. KRS indicates the second stage of processing rule data if the data passes the CND stage (DISPOSITION=P). The KRS (key resolution) stage verifies and resolves all foreign key relationships among tables that are involved in the import. XFR indicates the third stage of processing rule data. This stage transfers the rule data to the CZ online tables. OK indicates that the rule has been successfully imported. This is the final reporting stage. ERR indicates that the rule failed parsing. This is the

Field Name	Type	Description
		final reporting stage. For additional rule import information, see Rule Import, page 5-21.

Online Data Fields

The import tables' data fields exactly match the fields in the corresponding online table and are used to hold the data to be put into the online table.

Surrogate Key Fields

Surrogate key fields in the import tables hold the customer-provided extrinsic identifications for data to be imported. These include both foreign surrogate keys and surrogate primary keys.

Foreign Surrogate Key – A foreign surrogate key is a reference to a different table made through that table's surrogate primary key rather than through the online table's integer key value. A foreign surrogate key consists of one or more fields that resolve references from one import table to another. These keys are named FSK_*table_refno_fldnum*, where *table* is the name of the referenced table, *refno* is the number of the table-to-table reference, and *fldnum* is the position of the referenced surrogate-key field in the referenced import table. Note that *refno* is required to keep unique names for tables with multiple references to the same table, and generally, the *fldnum* is 1.

Surrogate Primary Key – As a rule, imported tables contain a single field named ORIG_SYS_REF, which is used to hold the external value that uniquely identifies each record. In some cases, however, the online CZ table has a primary key consisting entirely of references to other tables. In this case, the surrogate primary key actually consists of the foreign surrogate keys that correspond to the native foreign keys in the online table.

Dependencies Among Import Tables

Dependencies among import tables must be heeded especially when custom importing single tables. Dependencies Among CZ Schema Import Tables, page 4-8, "Foreign Surrogate Key" lists the column in the import table whose value is dependent on the table listed in "Depends on". For example, the FSK_ITEMTYPE_1_1 column in CZ_IMP_ITEM_MASTER gets its value from CZ_IMP_ITEM_TYPE.NAME and helps in key resolution. FSK_ITEMTYPE_1_1 (default) is populated depending on the PK_USEEXPANSION indicator (0, 1, or 2) in CZ_XFR_TABLES. See Populating Import Tables, page 5-12 for the order in which the CZ_IMP tables are populated.

Note: Oracle recommends that limited usage of FSK_***_EXT columns as these columns will eventually be desupported.

A strong dependency means a value is required to successfully import that record. If Default is YES, then there is a default value in that column and import succeeds even if the dependency is strong and no value is imported. The following Dependencies Among CZ Schema Import Tables, page 4-8 lists the dependencies.

The following table shows the dependencies between the import tables.

Dependencies Among CZ Schema Import Tables

Import Table Name	Depends on	Foreign Surrogate Key	Type of dependency	Default
CZ_IMP_DEVL_PROJECT	CZ_IMP_INTL_TEXT.TEXT_STR	FSK_INTLTEXT_1_1	STRONG	NO
CZ_IMP_LOCALIZED_TEXTS	CZ_IMP_DEVL_PROJECT.ORIG_SYS_REF	FSK_DEVLPROJECT_1_1	STRONG	N/A
CZ_IMP_ITEM_MASTER	CZ_IMP_ITEM_TYPE.NAME	FSK_ITEMTYPE_1_1	STRONG	YES
CZ_IMP_ITEM_PROPERTY_VALUE	CZ_IMP_PROPERTY.NAME	FSK_PROPERTY_1_1	STRONG	NO
CZ_IMP_ITEM_PROPERTY_VALUE	CZ_IMP_ITEM_MASTER.REF_PART_NBR	FSK_ITEMMASTER_2_1	STRONG	NO
CZ_IMP_ITEM_TYPE	NO	NO	NO	NO
CZ_IMP_ITEM_TYPE_PROPERTY	CZ_IMP_ITEM_TYPE.NAME	FSK_ITEMTYPE_1_1	STRONG	NO
CZ_IMP_ITEM_TYPE_PROPERTY	CZ_IMP_PROPERTY.NAME	FSK_PROPERTY_2_1	STRONG	NO
CZ_IMP_PROPERTY	NO	NO	NO	NO

Import Table Name	Depends on	Foreign Surrogate Key	Type of dependency	Default
CZ_IMP_PS_NODES	CZ_IMP_INTL_TEXT.TEXT_STR	FSK_INTLTEXT_1_1	STRONG	NO
CZ_IMP_PS_NODES	CZ_IMP_ITEM_MASTER.ORIG_SYS_REF	FSK_ITEMMASTER_2_1	STRONG	NO
CZ_IMP_PS_NODES	CZ_IMP_PS_NODES.ORIG_SYS_REF	FSK_PSNODE_3_1	STRONG	N/A
CZ_IMP_PS_NODES	CZ_PS_NODES.PARENT_ID	FSK_PSNODE_4_1	STRONG	N/A
CZ_IMP_PS_NODES	CZ_IMP_DEVL_PROJECT.ORIG_SYS_REF	FSK_DEVLPROJECT_5_1	STRONG	NO
CZ_IMP_PS_NODES	CZ_MODEL_REF_EXPLS	FSK_EXPLNODE_1_1	STRONG	N/A
CZ_IMP_PS_NODES	CZ_PS_NODES.REFERENCE_ID	FSK_PSNODE_6_1	STRONG	NA/
CZ_IMP_PS_NODES	CZ_EFFECTIVITY_SETS.EFFECTIVITY_SET_ID	FSK_EFFSET_7_1	STRONG	N/A
CZ_IMP_PS_NODES	SRC_APPLICATION_ID	FSK_ITEMMASTER_2_2	STRONG	N/A
CZ_IMP_PS_NODES	CZ_IMP_DEVL_PROJECT.ORIG_SYS_REF	FSK_DEVLPROJECT_5_1	STRONG	N/A

Control Tables

The control tables provide the mechanism for controlling what data is imported or refreshed when populating the CZ schema import tables with data from outside sources. The control table names are prefixed with CZ_XFR.

When running Oracle Configurator Populate and Refresh Configuration Models

Concurrent Programs, page C-18, records in the CZ_XFR tables determine which import tables are enabled for import, what data is imported, and how the data is imported.

The following tables control the import process at the table and field level:

- CZ_XFR_FIELDS
- CZ_XFR_PROJECT_BILLS
- CZ_XFR_TABLES

The following tables contain import information:

- CZ_XFR_RUN_INFOS
- CZ_XFR_RUN_RESULTS
- CZ_XFR_STATUS_CODES

CZ_XFR_TABLES identifies the mapping of the import table to the online table, as well as the rules for importing data into the CZ schema.

CZ_XFR_FIELDS identifies the transfer rules for the fields that are transferred during the Populate or Refresh Configuration Models concurrent programs. Every field is updated during import or refresh, but the update can be retracted by using the NOUPDATE flag in the CZ_XFR_FIELDS table. If a field that is transferred does not have an entry in the CZ_XFR_FIELDS table, then that field is updated.

For example, setting the NOUPDATE flag to 1 in the CZ_XFR_FIELDS table for CZ_ITEM_MASTERS.DESC_TEXT, inhibits the updating of the Item Master description in CZ_ITEM_MASTERS.DESC_TEXT when a Model is refreshed. Setting a value in the CZ_XFR_FIELDS Table, page 4-10 shows how to set the field in the CZ_XFR_FIELDS table so that changes made to the BOM Model's Item description do not appear in Oracle Configurator Developer.

Setting a value in the CZ_XFR_FIELDS Table

```
SQL> UPDATE CZ_XFR_FIELDS
     SET NOUPDATE = '1'
     WHERE order_seq = 4
     AND dst_field IN ('DESC_TEXT', 'REF_PART_NBR');

SQL> COMMIT
```

CZ_DB_SETTINGS Table

The CZ_DB_SETTINGS table provides parameters that affect certain applications and CZ schema processes.

Only one CZ_DB_SETTINGS table exists in a CZ schema.

Accessing the CZ_DB_SETTINGS Table

A user's responsibility determines whether they can view or edit the CZ_DB_SETTINGS table. A user must have the Oracle Configurator Administrator responsibility to edit the CZ_DB_SETTINGS table through concurrent programs. For more information, see View Configurator Parameters , page C-2 and Modify Configurator Parameters, page C-3.

Organization of the CZ_DB_SETTINGS Table

The parameters in the CZ_DB_SETTINGS table are mapped to a particular section of the CZ schema. The particular section is identified in the SECTION_NAME field and contains relevant database parameters. The sections are:

- IMPORT - Controls how BOM Model data is imported into the CZ schema
- LogicGen - Governs how the Model's logic is generated
- ORAAPPS_INTEGRATE - Controls how Oracle Configurator integrates with other Oracle Applications
- SCHEMA - Sets general parameters that control the CZ schema
- UISERVER - Governs the behavior of the runtime Oracle Configurator user interface

Each parameter contains the following fields:

- DATA_TYPE specifies the parameter's datatype. All CZ_DB_SETTINGS values are stored as VARCHAR2(255) in the VALUE field. If the DATA_TYPE is an integer, then the Configurator converts the data in the VALUE field to an integer before using it. For example, the Batchsize default value is stored as string 10000, but Configurator interprets string 10000 as integer 10000.
- SETTING_ID identifies the parameter.
- VALUE is the parameter's data. This value may be set during an installation or upgrade of the database instance. The Oracle Configurator Administrator can modify a value by running the Modify Configurator Parameters, page C-3 concurrent program.

CZ_DB_SETTINGS Parameters

Some of the CZ_DB_SETTINGS parameter values are predefined during an installation or upgrade of Oracle Configurator. The Oracle Configurator Administrator can modify the values of these parameters by running the Modify Configurator Parameters, page C-3 concurrent program. For information on running concurrent programs, see Running Configurator Concurrent Programs, page B-2. For specific information on modifying

the parameters in the CZ_DB_SETTINGS table, see Settings in CZ_DB_SETTINGS Table, page 4-12 that lists the parameters in the CZ_DB_SETTINGS table that can be modified.

Settings in CZ_DB_SETTINGS Table

SETTING_ID	SECTION_NAME	DATA_TYPE	Default VALUE	More information in...
AltBatchValidateURL	ORAAPPS_INTEGRATE	string	n/a	AltBatchValidateURL, page 4-15
BadItemPropertyValue	IMPORT	T/F	F	BadItemPropertyValue, page 4-16
BatchSize	SCHEMA	string	10000	BatchSize, page 4-16
BOM_REVISION	ORAAPPS_INTEGRATE	string	n/a	BOM_REVISION, page 4-17
CommitSize	IMPORT	integer	500	CommitSize, page 4-17
DISPLAY_INSTANCE_NAME	UISERVER	string	n/a	DISPLAY_INSTANCE_NAME, page 4-17
FREEZE_REVISION	SCHEMA	string	System setting	FREEZE_REVISION, page 4-18
GenerateGatedCombo	LogicGen	YES/NO	YES	GenerateGatedCombo, page 4-18
GenerateUpdatedOnly	LogicGen	YES/NO	YES	GenerateUpdatedOnly, page 4-18

SETTING_ID	SECTION_NAME	DATA_TYPE	Default VALUE	More information in...
GenStatisticsBOM	IMPORT	YES/NO	NO	GenStatisticsBOM, page 4-18
GenStatisticsCZ	IMPORT	YES/NO	NO	GenStatisticsCZ, page 4-18
MAJOR_VERSION	SCHEMA	integer	System setting	MAJOR_VERSION, page 4-18
MaximumErrors	IMPORT	integer	10000	MaximumErrors, page 4-18
MemoryBulkSize	IMPORT	integer	50000	MemoryBulkSize, page 4-19
MINOR_VERSION	SCHEMA	string	System setting	MINOR_VERSION, page 4-19
MULTISESSION	IMPORT	integer	0	MULTISESSION, page 4-19
OracleSequenceIncr	SCHEMA	integer	20	OracleSequenceIncr, page 4-19
PsNodeName	ORAAPPS_INTEGRATE	string	RefPartNbr	PsNodeName, page 4-20
PublicationLogging	ORAAPPS_INTEGRATE	YES/NO	NO	PublicationLogging, page 4-20

SETTING_ID	SECTION_NAME	DATA_TYPE	Default VALUE	More information in...
PublishingCopyRules	ORAAPPS_INTEGRATE	YES/NO	YES	PublishingCopyRules, page 4-20
PublicationLocalBOMSynch	PUBLICATION	YES/NO	NO	PublicationLocalBOMSynch, page 4-20
PurgeDeleteConfigBatchsize	SCHEMA	integer	100	PurgeDeleteConfigBatchsize, page 4-21
RefPartNbr	ORAAPPS_INTEGRATE	string	CONCATENATED_SEGMENTS	RefPartNbr, page 4-21
ResolvePropertyDataType	ORAAPPS_INTEGRATE	YES/NO	1-integer, 2-decimal, 3-boolean, 4-text	ResolvePropertyDataType, page 4-22
RestoredConfigDefaultModelLookupDate	ORAAPPS_INTEGRATE	string	config_creation_date	RestoredConfigDefaultModelLookupDate, page 4-23
Revision Date/User	SCHEMA	any string	-	Revision Date and User, page 4-23
RUN_BILL_EXPLODER	ORAAPPS_INTEGRATE	YES/NO	YES	RUN_BILL_EXPLODER, page 4-23
SuppressSuccessMessage	UISERVER	YES/NO	NO	SuppressSuccessMessage, page 4-24

SETTING_ID	SECTION_NAME	DATA_TYPE	Default VALUE	More information in...
TimeImport	IMPORT	string		TimeImport, page 4-24
UI_NODE_NAME_CONCAT_CHARS	ORAAPPS_INTEGRATE	string	n/a	UI_NODE_NAME_CONCAT_CHARS, page 4-24
UseLocalTableInExtractionViews	IMPORT	YES/NO	NO	UseLocalTableInExtractionViews, page 4-25
UtlHttpTransferTimeout	SCHEMA	integer	n/a	UtlHttpTransferTimeout, page 4-25

AltBatchValidateURL

AltBatchValidateURL allows the batch validation process to bypass the URL that is normally used for batch validation. This might be necessary if your database cannot communicate with your Web server.

If Oracle Configurator uses Secure Sockets Layer (SSL), then you can enable batch validation by creating an additional non-SSL-enabled (HTTP) servlet port and specifying its URL as the value of AltBatchValidateURL. For additional SSL information, see the Oracle Support Web site.

If your configurator servlet is set up to use HTTPS, then you can set AltBatchValidateURL to be a servlet using HTTP, and avoid some SSL encryption and handshaking overhead. Since the communication is between the Application database and the Application middle tier, communication does not cross the internet and thus HTTPS may not be necessary. This configuration does require extra setup and is not required.

You can also enable batch validation by using your existing SSL-enabled port for Oracle Configurator and setting up the Oracle Wallet for use by Oracle Configurator as described on the Oracle Support Web site. In this case, you do *not* set any value for AltBatchValidateURL.

If you use a firewall, have your database setup in a DMZ, or have some other network configuration where the database cannot communicate with the web server, then you

should set up an internal web server. After setting up an internal web server, you must then set the AltBatchValidateURL setting in the CZ_DB_SETTINGS table to be the URL for the configurator servlet on your internal web server.

For more information regarding DMZ setup, see the Oracle Support Network site.

To insert the AltBatchValidateURL into the CZ_DB_SETTINGS table, use the SQL INSERT statement shown in Adding AltBatchValidateURL to CZ_DB_SETTINGS, page 4-16.

Adding AltBatchValidateURL to CZ_DB_SETTINGS

```
INSERT INTO cz_db_settings (setting_id, section_name, data_type, value,
desc_text) VALUES
('AltBatchValidateURL','ORAAPPS_INTEGRATE',4,'http://servername.com:8808
/OA_HTML/UiServlet','Non-secure URL')
```

BadItemPropertyValue

BadItemPropertyValue indicates the action that is taken when an Item's PROPERTY _VALUE in the CZ_IMP_ITEM_PROPERTY_VALUES table does not match the DATA_TYPE in the CZ_PROPERTIES online table. The default value (F) forces the record to be updated to include the PROPERTY_VALUE so that it is imported into the CZ_ITEM_PROPERTY_VALUES online table. Valid Values for the BadItemPropertyValue Setting, page 4-16 lists the valid values for BadItemPropertyValue setting and the disposition:

The following table lists the valid values for the BadItemProperty value setting, along with a description of the value.

Valid Values for the BadItemPropertyValue Setting

Value	Disposition
R	Reject the record in the import table and use the old PROPERTY_VALUE
F	Force the record to be updated to include the PROPERTY_VALUE from the import table
K	Update all information in the record except the Item PROPERTY_VALUE
X	Reject the record and logically delete any matching Item property value record in the CZ_ITEM_PROPERTY_VALUES table. The Item property value defaults to the property default value in the CZ_ITEM_PROPERTY_VALUES table.

BatchSize

BatchSize indicates the number of records that are modified before committing a

transaction in batch operations. The BatchSize setting is also used during a purge operation.

Ordinarily a database stored procedure runs as a single transaction that is considered pending until the calling operation commits the transaction. The pending changes are lined up in a rollback segment. If the calling operation is cancelled, then the transaction is rolled back. If the calling operation encounters an error, then the pending changes in the rollback segment are discarded. However, some batch operations, such as import, can involve many more records than the database can handle as a single transaction. If the transaction is too big, then the database fails an operation with a rollback-segment error. To avoid a rollback_segment error, import and other batch-like operations count up the modified records in the database and when the count matches the BatchSize value, the operation commits the transaction and resets the counter. Every record is not committed individually because it is considerably more economical to commit many updates at once.

BOM_REVISION

BOM_REVISION indicates the BOM revision in the Oracle Applications database from which data is being imported into the CZ schema. This setting is checked to ensure that the correct date format is used in the call to the BOM Model explosion procedure.

The value of BOM_REVISION is the Oracle Applications revision number used to determine which explosion date format to use. . Valid values are `5.0.628` for Release 10.7, `11.0.28` for Release 11.0, and `11.5.0` for both Release 11*i* and Release 12. If the value is null (default), then 11.5.0 is used. The call to the BOM Model explosion procedure checks up to the second decimal point of this value.

If the value is 11.5.*n* or Release 12, then the date format YYYY-MM-DD is used. Otherwise, DD/MON/RR is used for Release 10.7 or 11.0.

CommitSize

CommitSize indicates the number of import records in each database transaction between commits. CommitSize has the same purpose as BatchSize. for more information, see BatchSize, page 4-16. CommitSize is used during import.

DISPLAY_INSTANCE_NAME

DISPLAY_INSTANCE_NAME determines whether an Instance Name column appears in the Oracle Configurator Summary page. Oracle Configurator checks this setting only if multiple instances of one or more components exist in the configuration.

If DISPLAY_INSTANCE_NAME is set to `TRUE`*and* at least one component in the configuration has multiple instances, then the Instance Name column appears and displays the name of each instance.

If DISPLAY_INSTANCE_NAME is set to `FALSE` or there are no components with multiple instances in the configuration, then the Instance Name column does not appear. If set to `False` but there are multiple instances in the configuration, then

instance names appear in the Description column (instead of each Item's description).

FREEZE_REVISION

FREEZE_REVISION indicates the revision number at the freeze stage. This parameter is used to capture the revision levels for the implementation of database package bodies and views. For example, if a table is tuned to improve performance, but the fields and the data returned are the same, then there is no need to change the MAJOR_VERSION or MINOR_VERSION but the FREEZE_REVISION value reflects the reworked view. This setting is read-only and populated when applying a patch.

GenerateGatedCombo

GenerateGatedCombo determines how a `FALSE` logic state is propagated in Explicit Compatibility, Property-based Compatibility and Design Chart Rules. See the *Oracle Configurator Developer User's Guide* for additional information about Gated Combinations.

GenerateUpdatedOnly

GenerateUpdatedOnly set to `YES`, causes logic generation to skip all referenced Models whose logic is up-to-date. GenerateUpdatedOnly set to `NO` causes the logic of all referenced Models to be generated even if their logic is up-to-date.

GenStatisticsBOM

GenStatisticsBOM set to `YES` forces the optimizer to update the internal statistics on the BOM_EXPLOSIONS table before running queries in the CZ schema. Generating statistics allows the optimizer to choose a better execution plan based on the current data structure in a table.

GenStatisticsCZ

GenStatisticsCZ set to `YES` forces the optimizer to update the internal statistics on the entire CZ schema before running queries in the CZ schema. Generating statistics allows the optimizer to choose a better execution plan based on the current data structure in a table.

MAJOR_VERSION

MAJOR_VERSION indicates the major version label for the CZ schema. This setting is read-only and is populated when upgrading the schema.

MaximumErrors

MaximumErrors indicates the limit of errors allowed before an import run is terminated. If you have a large amount of data to import, or you are not concerned with the process stopping once a certain number of errors is reached, then set this parameter

to an extremely large number.

MemoryBulkSize

MemoryBulkSize regulates the memory usage of import. The smaller the setting, the less memory is required for import. This number is used during import for the `cz_ps_nodes` extraction procedure for specifying the number of records that are processed in the same pass. If the value entered is less than the total number of records to be imported, then the specified number of records is loaded and processed, and then the next group of records is loaded and processed. If there is no value entered, then the MemoryBulkSize is set to 10000000.

MINOR_VERSION

MINOR_VERSION indicates the minor version label for the CZ schema. This value is read-only and is populated when applying a patch. The MINOR_VERSION does not change during a particular family pack release.

MULTISESSION

MULTISESSION indicates the way in which a new import session interacts with other import sessions.

- A positive value indicates the number of seconds to wait while another import session is running. The current state is checked every second. After the number of seconds has elapsed, control goes to the waiting import session if no other session is active, or an exception is raised if another import session is still running.

- A value of `0` means do not wait if another import session is running, and immediately raise an exception if a session is already running.

- A negative value means ignore other import sessions and run this import session immediately without raising an exception. Setting this parameter to a negative number is equivalent to disabling it. If a session is currently running and a new import session begins, then the first session is not aborted and there is the risk of data corruption.

When MULTISESSION is missing from the CZ_DB_SETTINGS table, it is equivalent to the default 0.

If an import session is terminated, then the CZ_XFR_RUN_INFOS table may end up in an inconsistent state with the value of COMPLETED something other than 1.

OracleSequenceIncr

OracleSequenceIncr indicates the number of primary-key values allocated by each use of a sequence. The default setting means that keys are assigned in increments of `20`. Both runtime Oracle Configurator and Configurator Developer ask for a sequence value once, and then manage the sequence value minus 1 in memory. When the block is used

up, runtime Oracle Configurator and Configurator Developer again call for a sequence value. Keeping the default value at `20` saves round trips to the database.

> **Warning:** Changing the default OracleSequenceIncr setting of `20` is likely to have adverse effects. The value of OracleSequenceIncr should not be modified.

PsNodeName

PsNodeName indicates the source field to be loaded into the NAME field in the CZ_PS_NODES table. The source field is either the RefPartNbr or the DESCRIPTION field in the MTL_SYSTEM_ITEMS table. RefPartNbr is the default so that the name loaded into the Model structure in Oracle Configurator Developer matches the name in CZ_ITEM_MASTERS.

PublicationLocalBOMSynch

PublicationLocalBOMSynch controls whether the BOM Synchronization process runs automatically when a publication is created for a Model that was imported from a remote server, on the same instance in which Configurator Developer is running. If BOM Synchronization does not run in this scenario, it is possible for the publication lookup process to fail if the host application is also running on the local instance that launched Oracle Configurator.

To automatically run BOM Synchronization in the scenario described above, set the CZ_DB_SETTINGS parameter PublicationLocalBOMSynch to `YES` by running the Modify Configurator Parameters concurrent program. Enter the following parameters when submitting this program:

- Section Name: Publication
- Setting ID: PublicationLocalBOMSynch
- Value: YES
- Type: 4

PublicationLogging

PublicationLogging indicates whether a trace of the publication process is logged in the CZ_DB_LOGS table. The trace is helpful for debugging purposes and can be viewed in the log file. For more information about viewing log files, see Viewing Log Files, page B-4.

PublishingCopyRules

PublishingCopyRules indicates whether or not configuration rules are copied during publishing. If PublishingCopyRules is set to `NO`, then *only* Configurator Extension rules

are copied during publishing. The publishing process is faster when PublishingCopyRules is set to NO.

If the PublishingCopyRules is set to YES, then all rules are copied and both the source and published Models have the same rules.

> **Note:** Setting 'PublishingCopyRules' to 'NO' only affects you if changes are made to logic generation that are incompatible with previous versions of Oracle Configurator. If the rules for a published Model are not copied, then you cannot generate logic for the published Models. Using the NO setting requires republishing all published Models.

PurgeDeleteConfigBatchsize

When you run the concurrent program Purge Configurator Tables, page C-4, you can control its commit behavior by setting this parameter, which specifies how often the purge program issues a commit, in terms of a number of configurations. For example, a value of 200 specifies a commit after deleting a batch of 200 configurations.

RefPartNbr

RefPartNbr identifies the source fields that are loaded from the MTL_SYSTEM_ITEMS table into CZ_ITEM_MASTERS.REF_PART_NBR. This is a segment from the System Item key flexfield definition.

RefPartNbr determines what name is displayed for each imported Model structure node. The default value 'CONCATENATED_SEGMENTS' enables the BOM Model import process to construct BOM Model node names using multi-segment part numbers.

When RefPartNbr is set to 'SEGMENT1', only MTL_SYSTEM_ITEMS.SEGMENT1 is the source of the node names in the imported Model structure. If you want to use only the first segment of a part number as the node name, the Oracle Configurator Administrator must manually set RefPartNbr to 'SEGMENT1' by running the Modify Configurator Parameters concurrent program.

Any value for RefPartNbr other than 'CONCATENATED_SEGMENTS' or 'SEGMENT1' causes the import process to retrieve the value of the DESCRIPTION column from MTL_SYSTEM_ITEMS and displays the Item description as the node name in Configurator Developer.

> **Warning:** Examine MTL_SYSTEM_ITEMS_VL. CONCATENATED_SEGMENTS to verify that the field is correctly populated. If the field is incorrectly populated, then the entry in Oracle Inventory may be wrong. If the entry is correct, check CZ_IMP_ITEM_MASTER.REF_PART_NBR to see that the value is the same as that in MTL_SYSTEM_ITEMS_VL.

CONCATENATED_SEGMENTS.

Concatenated segments, including separators, must not exceed 1000 characters, which is the limit of the CZ_PS_NODES.NAME field. Any description longer than 1000 characters is truncated. The default separator is a dot (.). Other valid separators are |, -, or a custom value. See the *Oracle Inventory User's Guide* for more information about setting up part numbers.

You can enter multi-segment Items in the From Item and To Item input fields when you run either the Populate or Refresh Configuration Models concurrent program. You must include any separators that exist in the Item's part number when you enter multi-segment Item names.

> **Warning:** When updating an existing Model in Configurator Developer to use multi- segment part numbers, you must either reimport or refresh the BOM Model. Confirm that the BOM Model is getting re-exploded during import. The CZ_DB_SETTINGS.RUN_BILL_EXPLODER should be Yes.

ResolvePropertyDataType

ResolvePropertyDataType controls whether Item Catalog Descriptive Elements are imported into Configurator Developer as Item Properties with a data type of Text or Decimal Number. If the value for this setting is NO, all imported Item Properties have a data type of Text in Configurator Developer.

If the value of this setting is YES, then all Descriptive Elements whose value is a number are imported as Item Properties and have a data type of Decimal Number. All Descriptive Elements whose value is text (for example, Weight) have a data type of Text.

If ResolvePropertyDataType is null, then all Descriptive Elements are imported into Configurator Developer as Item Properties with a data type of Text.

ResolvePropertyDataType Setting , page 4-22 table illustrates how ResolvePropertyDataType affects how Descriptive Elements values are imported into Oracle Configurator Developer.

ResolvePropertyDataType Setting

ResolvePropertyDataType Setting	Item Catalog Descriptive Element Value	Data Type in Oracle Configurator Developer
YES	15	Decimal Number

ResolvePropertyDataType Setting	Item Catalog Descriptive Element Value	Data Type in Oracle Configurator Developer
YES	Length	Text
NO	15	Text
YES	Length	Text
null	'Length' or '15'	Text

Item Property is a protected field in the CZ schema (the NOUPDATE flag is set during import). Once you import a BOM Model, you cannot change an Item Property's data type simply by modifying the ResolvePropertyDataType setting and then refreshing the BOM Model.

RestoredConfigDefaultModelLookupDate

RestoredConfigDefaultModelLookupDate setting controls which publication Oracle Configurator uses on an order when called from Order Management. If this setting is config_creation_date, then Oracle Configurator uses the order line creation date. If this setting is null, then Oracle Configurator uses sysdate.

For more information, see DEFAULT_RESTORED_CFG_DATES, page 17-49.

Revision Date and User

Revision Date and User is read-only and documents the date and time at which the CZ schema was last upgraded, and the username of the user who performed the task.

RUN_BILL_EXPLODER

RUN_BILL_EXPLODER is a YES/NO flag (default=YES) that indicates whether the Oracle Applications **Bills of Material** exploder should be run on each bill that is marked for import in the CZ_XFR_PROJECT_BILLS table in the CZ schema at the time of import. See Populating the CZ Schema, page 5-1 for more information on exploding a BOM Model.

The Oracle Configurator Populate or Refresh Configuration Models concurrent programs load bills and Items based on top bills listed in the CZ_XFR_PROJECT_BILLS table in the CZ schema. Before extracting, if the RUN_BILL_EXPLODER setting is set to YES, then the procedure calls the BOM Model exploder to refresh data in BOM_EXPLOSIONS for each record in the CZ_XFR_PROJECT_BILLS table. If RUN_BILL_EXPLODER is set to NO, then the concurrent program transfers the BOM Models that are flagged for import in the CZ_XFR_PROJECT_BILLS table without running the BOM Model exploder first.

> **Note:** The Populate or Refresh Configuration Models concurrent programs do not explode BOM Models when importing from a remote server. See Exploding BOM Models in Oracle Applications, page 5-10 for details.

CZ_INTL_TEXTS contains the text string from the DESCRIPTION field in the BOM_EXPLOSIONS table for each imported BOM Model structure node.

The Oracle Configurator SQL*Plus scripts and concurrent programs target all or a subset of BOM Models exploded in the BOM_EXPLOSIONS table in the Oracle Applications database. Selected BOM Model Items come from the BOM_BILL_OF_MATERIAL and the BOM_INVENTORY_COMPONENTS tables.

> **Note:** Importing a BOM Model from a remote instance may fail if `RUN_BILL_EXPLODER` is set to `YES` in the local instance. (In this case, an error message similar to the following appears: "ORA-03113: end-of-file on communication channel.") If this occurs, set `RUN_BILL_EXPLODER` flag to `No` and then re-submit the import concurrent program. The new setting should enable the process to complete successfully.

SuppressSuccessMessage

The SuppressSuccessMessage setting affects runtime Oracle Configurator behavior by suppressing messages that would normally be shown. The setting determines whether a message is displayed after fixing a validation error.

If SuppressSuccessMessage is set to NO, then after fixing a validation error a runtime success message is displayed. If SuppressSuccessMessage is set to YES, then after fixing a validation error a runtime success message is not displayed.

To insert SuppressSucessMessage into CZ_DB_SETTINGS, use the SQL*Plus INSERT statement shown in Adding SuppressSuccessMessage to CZ_DB_SETTINGS, page 4-24 .

Adding SuppressSuccessMessage to CZ_DB_SETTINGS

```
INSERT INTO cz_db_settings (setting_id, section_name, data_type, value,
desc_text) VALUES ('SuppressSuccessMessage','UISERVER',4,'No','Runtime
display of success messageS')
```

TimeImport

TimeImport enables the collection of timing information during import.

UI_NODE_NAME_CONCAT_CHARS

UI_NODE_NAME_CONCAT_CHARS sets the concatenation character that is used when generating UI captions using both the node name and description. The default

concatenation character separating each text string is a comma surrounded by two spaces. (For example: "AT62431 , Sentinal Custom Laptop"). The Oracle Configurator Administrator can change the concatenation character that separates each string by running the Modify Configurator Parameters concurrent program.

UseLocalTableInExtractionViews

UseLocalTableInExtractionViews is a YES/NO flag. If UseLocalTableInExtractionViews is set to YES, then definitions of some import extraction views include the DUAL table in the join. The UseLocalTableInExtractionViews setting is ignored if the import source server is local.

> **Note:** If you are importing or refreshing from a remote database instance and the database instance is version 8*i*, then UseLocalTableInExtractionViews must be set to `YES`. This is because of an RDBMS bug. If this setting is not YES, then the following error appears in the cz_db_logs table after running the Populate and Refresh Configuration Models Concurrent Programs, page C-18 :
> `"ORA-01025: UPI parameter out of range"`

UtlHttpTransferTimeout

UtlHttpTransferTimeout allows modification of the timeout length that is used inside the call to the UTL_HTTP.REQUEST procedure during batch validation. The value is the number of seconds. Once the call completes, the timeout is set back to its original value.

To insert UtlHttpTransferTimeout into the CZ_DB_SETTINGS, use the SQL*Plus INSERT statement shown in Adding UtlHttpTransferTimeout to CZ_DB_SETTINGS, page 4-25.

Adding UtlHttpTransferTimeout to CZ_DB_SETTINGS

```
INSERT INTO cz_db_settings (section_name, setting_id, data_type, value,
desc_text)
SELECT 'SCHEMA', 'UtlHttpTransferTimeout', 1, '60', 'HTTP timeout for
batch validation'
FROM DUAL WHERE NOT EXISTS
(SELECT NULL FROM cz_db_settings
WHERE section_name='SCHEMA'
AND upper(setting_id)='UTLHTTPTRANSFERTIMEOUT');
```

> **Note:** This functionality is available only in Oracle 9*i* and later.

5

Populating the CZ Schema

This chapter provides an overview of why and how to import data from Oracle Applications and non-Oracle Applications databases. It describes the import processes, the import tables used during data import, how to import data into the CZ schema, data import verification, the process for refreshing or updating imported data, and customizing data import.

This chapter covers the following topics:

- Overview
- Introduction
- Standard Import
- Rule Import
- Rule Import Procedure
- Custom Import
- Custom Import Procedure

Overview

This chapter provides an overview of why and how to import data from Oracle Applications and non-Oracle Applications databases. It describes the import processes, the import tables used during data import, how to import data into the CZ schema, data import verification, the process for refreshing or updating imported data, and customizing data import. The import processes discussed are:

- Standard Import, page 5-4
- Rule Import, page 5-21
- Custom Import, page 5-30

For information about the CZ schema, see the CZ Schema, page 4-1 chapter.

Introduction

Populating the CZ schema usually begins by importing data. There are three types of data import:

- Standard import of Oracle Applications **BOM** Models and Inventory data into the CZ schema. For more information, see Standard Import, page 5-4.
- Rule import of legacy rules written in Constraint Definition Language (CDL) format into the CZ schema. For more information, see Rule Import, page 5-21.
- Custom import of data that is not handled by a standard import. For more information, see Custom Import, page 5-30.

Once the CZ schema is populated with imported data, that data is then available in Oracle Configurator Developer and the runtime Oracle Configurator.

This section lists:

- Types of Data Stored in the CZ Schema During Development and Runtime, page 5-2
- Means of Populating the CZ Schema, page 5-3
- CZ_IMP Tables, page 5-4

Types of Data Stored in the CZ Schema During Development and Runtime

The data stored in the CZ schema includes:

- Configuration models:
  - Item and Model structure data
  - Configuration rules
  - Customized User Interface (UI) Templates
  - UI definitions
  - Publication records
- Configurations
- Configurator Extension Archives
- Oracle Configurator system settings

- Oracle Configurator transfer information

See Means of Populating the CZ Schema, page 5-3 for information on how this data is inserted. See Standard Import, page 5-4 for more details about the specific kinds of Inventory and BOM Model data stored in the CZ schema.

Means of Populating the CZ Schema

The CZ schema is populated with data by the following means:

- Concurrent programs in Oracle Applications import Item and Model structure data from outside sources into the CZ schema. For more information on preparing data for import, see Preparing the Data for Import, page 5-7. For more information, see Populate and Refresh Configuration Models Concurrent Programs, page C-18.

  Note: When you submit an Oracle Applications concurrent request to populate and refresh Models, the Model, any referenced Models, and any referenced UI Content Templates must either be unlocked or locked by you. For more information on locking, see the *Oracle Configurator Developer User's Guide*.

- A concurrent program in Oracle Applications imports rules written in CDL format into the CZ Schema. These rules may be legacy rules that are rewritten in CDL. For more information on preparing rules for import, see Rule Import Procedure, page 5-29. For more information about the concurrent program, see Import Configuration Rules, page C-23.

- Custom programs load data transfer files into the CZ schema. For more information see Identifying Data for a Custom Data Import. , page 5-31

- Concurrent programs migrate Item and Model structure data from one CZ schema into another CZ schema. For more information, see Migrating Data, page 6-1 and Migration Concurrent Programs, page C-30.

- Configurator Extensions populate CZ table fields with configuration data that cannot be directly inserted using the runtime Oracle Configurator. For more information, see the *Oracle Configurator Extensions and Interface Object Developer's Guide*, and Migrate Functional Companions, page C-32.

- End users select certain nodes of configuration models that pass configuration attributes to the CZ schema. For more information, see the *Oracle Configurator Methodologies* documentation.

- Oracle Configurator Developer populates the CZ schema with configuration model data, including rule, publishing, and UI definitions. For more information on the information in the CZ schema, see the Oracle Integration Repository.

- Programmatic tools used to develop and maintain configuration models, and deploy a runtime Oracle Configurator populate the CZ schema. For more information, see Programmatic Tools for Development, page 17-1 and Programmatic Tools for Maintenance, page 18-1.

CZ_IMP Tables

The CZ_IMP tables store imported data and keep track of the success or failure when importing data into the CZ schema. The CZ_IMP tables correspond to the equivalent CZ online tables. The imported data becomes available to Configurator Developer when the Populate and Refresh Configuration Models Concurrent Programs, page C-18 or Execute Populators in Model Concurrent Program, page C-29, or a custom import moves the data from the import tables into the corresponding online tables. Configurator Developer and the runtime Oracle Configurator read the imported data from the CZ online tables.

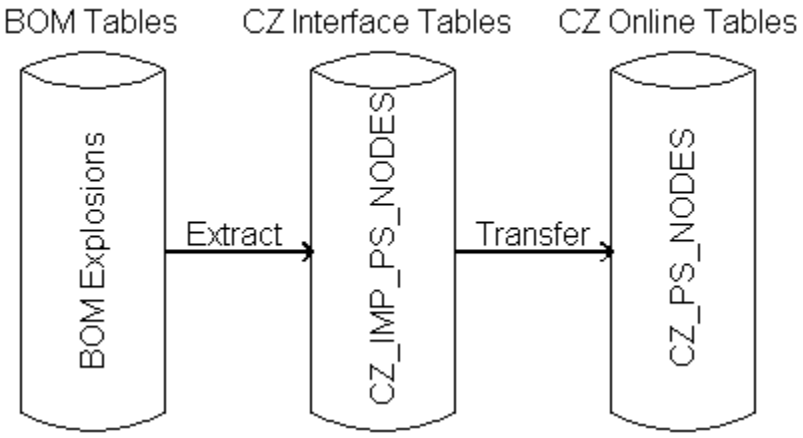
For example, when an Item in the CZ_ITEM_MASTERS table is imported into the CZ schema, the Item data also appears in the CZ_IMP_ITEM_MASTER table. For a list of tables that store imported data, see ITEM Item-Master Tables, page D-2. For more information about where various kinds of data are stored in the Oracle Integration Repository.

Standard Import

A standard import consists of transferring data from Oracle Applications **Bills of Material** (Releases 10.7, 11.0, 11*i*, or Release 12) to Oracle Configurator Release 12. Data Flow in the Import Process, page 5-5 shows the data flow when importing a BOM Model.

Data Flow in the Import Process, page 5-5 shows the flow of data during import. The data is extracted from the BOM tables such as BOM_EXPLOSIONS into the CZ Interface tables such as CZ_IMP_PS_NODES and then transferred to the CZ Online tables such as CZ_PS_NODES.

Data Flow in the Import Process



When developing a configuration model, Oracle Configurator Developer accesses the CZ schema, not the Oracle Applications Inventory and Bills of Material schemas. However, when ordering Items that have been configured based on a configuration model, the runtime Oracle Configurator accesses the CZ schema.

The CZ schema must contain an exact replication of the BOM Model's structure, rules and Item data. This exact replication is necessary to create configurations of BOM Models that participate in downstream processes such as ordering.

This standard import section describes:

Inventory and BOM Data That Can Be Imported

A standard import involves importing Oracle Applications Inventory and BOM Model data into the CZ schema. Specifically, the imported data is:

- Bills of Material structure (ATO and PTO BOM Models)
- Inventory data
- ATO or PTO BOM Model rules:
  - Optional or required
  - Minimum and maximum quantity
  - Mutually exclusive
  - Quantity cascade
- Attributes in Oracle Inventory such as Item Catalog Group, Catalog Descriptive Elements and values

Overall Standard Import Procedure

The overall procedure for a standard import is:

1. Determine the import source and target (see Database Instances, page 3-1)
2. Prepare the data (see Preparing the Data for Import, page 5-7).
3. If the import source is a remote database:
   1. The Configurator Administrator must define and enable the source server for import (see Defining and Enabling a Server for Import, page 5-10).
   2. Explode the BOM Models that you want to import (see Exploding BOM Models in Oracle Applications, page 5-10).
4. Optionally identify specific data to be ignored during the import (see Controlling the Data for Import, page 5-11).
5. Run the Populate and Refresh Configuration Models Concurrent Programs, page C-18 in Oracle Applications to import the BOM Model's data into the CZ Schema.
6. Verify that the data import succeeded (see Verifying the Data Import, page 5-16).
7. If you re-import the same BOM Model from a different source, you must first

synchronize your BOM-based configuration models with the new source (see Synchronizing Data, page 7-1).

8. Because repeated data imports can result in large amounts of logically-deleted Items in the CZ schema, run the Purge Configurator Tables, page C-4 concurrent programs to improve database performance. For more information, see Purging Configurator Tables, page 8-2.

Determining the Import Data Source Instance and the Target Instance

The source of imported data is also called the import source or remote server. The import source should be a production database. Oracle Configurator supports importing BOM Model data from only one Oracle Applications database. This is because the information used to refresh imported Oracle Applications BOM Models can overlap among multiple Applications databases. See Defining and Enabling a Server for Import, page 5-10 for information about changing the import source.

You cannot test a published BOM on a local server if the BOM is defined remotely. When you publish a Model, synchronization takes place. If a Model is published locally but the source BOM is defined remotely, then synchronization does not occur.

The target of the imported data is the database instance you have designated for developing your BOM-based configuration model. You run the Populate and Refresh Configuration Models Concurrent Programs, page C-18 in Oracle Applications in the target database instance.

For more information about selecting or changing which database instance should serve as import source and which should be the target, see Database Instances, page 3-1.

Preparing the Data for Import

For purposes of consistency with other processes in your business, use production data. Preparing the data for standard import involves creating a BOM Model using Oracle Inventory Items. Only Oracle Inventory Items that are associated with a BOM Model in Oracle Bills of Material can be imported into the CZ schema. If you are importing other data or data from non-Oracle Applications databases, see Custom Import, page 5-30. If you are importing rule data from non-Oracle Applications databases or standalone rules, see Rule Import, page 5-21.

Determine which version of Oracle Applications is the import source. You can import BOM Models only from Release 10.7, 11.0 and 11i to Release 12. Standard import requires that BOM Models be complete and identified at the top level. Identifying the BOM Model at the top level insures that all child BOM Models are imported. If a BOM Model is not complete, then a warning message is displayed. For information on importing BOM Models with child BOM Models, and BOM Models with a Common Bill, see BOM Model with a Common Bill, page 5-21.

Note: Items for standard import must be defined in Oracle Applications Inventory and then specified for inclusion in a BOM Model in Oracle Bills of Materials.

To create a BOM Model in Oracle Applications, you must first define the Items (see Defining Inventory Items for Configuration, page 5-8) and then their hierarchical relationship in a BOM Model (see Creating BOM Models for Configuration, page 5-9).

Defining Inventory Items for Configuration

Begin data preparation by defining Inventory Items that can be used to build a BOM Model and provide the Item data needed for implementing a configuration model.

If you are using Multiple Language Support (MLS), you should enter translated descriptions of BOM Model Items before importing data to the CZ schema. See Multiple Language Support, page 14-1.

In Oracle Applications Inventory:

- Define the Items of your BOM Model and specify a **BOM Item Type** of Standard, Option Class, or Model for each Item.
- Select the **Inventory Item** check box to make each Item both configurable and orderable.
- Select the **BOM Allowed** check box if the Item can be assigned as a component on a BOM Model or can be used to create a BOM Model.
- Assign **Item Catalog Groups** and **Descriptive Elements** to Items for which you want imported Properties in Configurator Developer.
- Indicate whether the Items that you want to be a BOM Model are a **Pick To Order** (PTO) or **Assemble To Order** (ATO).
- Select the **OM Indivisible** check box if Item quantities should be treated as integers (see Importing Decimal or Integer Quantities, page 5-13).

BOM Item Type determines whether an Item can be a component in a bill of materials, may contain child components, or can also be a BOM Model. A BOM Option Class typically contains one or more Standard Items. See Importing Decimal or Integer Quantities, page 5-13 for details about importing Standard Item quantities as integers or decimals. For more information on Standard Items, see the *Oracle Bills of Material User's Guide*.

Any Item that is defined as a Model in Oracle Inventory and exists as a component in another BOM Model (for example, a PTO BOM Model that contains an ATO BOM Model), must also be defined as a BOM Model in Oracle Bills of Material to be imported into the CZ schema.

When an Item is a component of a PTO or ATO BOM Model and at the same time is the parent of other component Items, the **BOM Allowed** check box must be selected for that Item. When a Standard Item is defined this way, it can be a "kit" containing other Standard Items. Standard Items included in a kit are always required (mandatory); they are never optional. The **BOM Allowed** check box must be selected for all of the component Items within the kit.

Item Catalog **Descriptive Element** values do not have a data type in Oracle Inventory. When you import BOM Model data into the CZ schema, Descriptive Elements become Item Properties. These Item Properties have a data type of Text, or Decimal Number.

By default, the Descriptive Element's value is imported as a decimal number if the value is a number; otherwise, the value is imported as text. However, you can modify how these values are imported using the ResolvePropertyDataType setting in the CZ_DB_SETTINGS table. For details, see ResolvePropertyDataType, page 4-22.

For more information about imported BOM Models and Properties, see the *Oracle Configurator Developer User's Guide*.

For more information about defining Items, see the *Oracle Inventory User's Guide*.

Creating BOM Models for Configuration

After defining Inventory Items, you must continue in Bills of Material to create the BOM Model.

- Select an Inventory Item that has a BOM Item Type of Model, and add other BOM Models, Option Class Items, and Standard Items as components within the BOM Model.

- In a multiple organization supply chain implementation, set the Item attributes **Check ATP** and **ATP Components** to control the extent of the search made by Global Order Promising for available-to-promise inventory.

  For more information about the **Check ATP** and **ATP Components** settings, see the *Oracle Advanced Supply Chain Planning and Oracle Global ATP Server User's Guide*.

- Specify attributes for each component in the bill, such as whether a BOM Model or BOM Option Class contains **Mutually Exclusive** Items and whether the component is required.

When the **Mutually Exclusive** option is selected, the optional child components of that Option Class mutually exclude one another based on the minimum and maximum number of components allowed in a valid configuration.

Required Items do not participate in the configuration process and therefore are not imported into the CZ schema. (An exception is when a required component contains optional components; in this case, it *is* imported into the CZ schema). Required Items are added automatically to the configured work order by the AutoCreate Configuration Items concurrent program.

For more information about creating a BOM Models, see the *Oracle Bills of Material User's Guide*.

Defining and Enabling a Server for Import

The local database instance is the default import server, meaning if you do not specifically enable a server for import, the database instance in which you run the import is used as the source.

If you are transferring data to the CZ schema from a Bills of Material schema in a different database instance, you must define that import source as a remote server. See Server Administration, page B-3 for information about defining and enabling a remote server. Several servers can be defined and enabled, but only one server is Import Enabled.

If you need to define and enable a remote server for import, you must first submit a Modify Server Definition, page C-14 concurrent request to disable the local server for import, and then define and enable the remote server where the import source data is stored.

Oracle requires that you define only one server for import. If an import server is changed after BOM Models have been imported, then the configuration models must be synchronized to the BOM Models on the new import server. For details on synchronizing the configuration models with the BOM Models on the newly defined remote server, see BOM Model Synchronization Process, page 7-2.

Exploding BOM Models in Oracle Applications

Prior to importing or refreshing a BOM Model into the CZ schema from Bills of Material (Releases 10.7, 11.0, 11i, or 12) in another instance (remote server), you must explode the BOM Model.

The following sections explain how to explode a BOM Model in different releases of Oracle Applications.

Exploding a BOM Model in Release 12

To explode a BOM Model in Oracle Applications, Release 12:

1. Log in to Oracle Applications using the appropriate username and password.
2. Select the **Order Management** responsibility.
3. Select **Orders, Returns > Sales Orders**.
4. Enter all required data in the **Main** tabbed region.
5. Click the **Line Items** tabbed region.

6. On the Order Line, select the root Model that you want to import into Oracle Configurator from the Item list of values. This is the same Model that you select when creating a new object in Oracle Configurator Developer or running the Populate Configuration Models, page C-19 concurrent program in Oracle Applications.

   The BOM Model explosion process is called recursively for as many levels as necessary in the root Model.

7. Enter 1 in the **Qty** field, then click **Configurator**.

8. After all the BOM Model's components are displayed, click **Cancel** to close the Configurator page.

Exploding a BOM Model in Release 10.7 or 11.0

To explode a BOM Model in Oracle Applications, Release 10.7 or 11.0:

1. Log in to Oracle Applications using the appropriate username and password.

2. Select the **Order Entry** responsibility.

3. Navigate to the Sales Orders page, enter all required fields.

4. On the Order Line, select the Model that you want to import into Oracle Configurator from the Item list of values. This is the same Model that you select when creating a new object in Oracle Configurator Developer or running the Populate Configuration Models, page C-19 concurrent program in Oracle Applications.

5. Enter 1 in the **Qty** field, then click **Configurator**.

6. After all the BOM Model's components are displayed, select **Cancel** to close the Configurator page.

7. Repeat steps 1 through 6 for each BOM Model that you want to import into the CZ schema.

Controlling the Data for Import

Controlling data import involves identifying or customizing what data gets imported.

To do this you run concurrent programs to set the values in the CZ_XFR_ control tables in the CZ schema that control import. See Control Tables, page 4-9 for more information about the control tables. See Importing the Data, page 5-15 for information about identifying what data gets imported.

Importing Data Into Specific Tables

When you import data, you must be aware of the dependencies between the import tables. For more information, see Dependencies Among CZ Schema Import Tables, page 4-8.

You may want to specify only a group of tables from which extracted data is loaded into the import tables. The CZ_XFR_TABLES.DISABLED field determines whether a specific table is enabled or disabled for import.

For general information on running concurrent programs, see Running Configurator Concurrent Programs, page B-2. For details on importing data into specific tables, see Select Tables to be Imported, page C-42.

In Oracle Applications, you can also display the current tables to be imported by selecting the concurrent program, Show Tables to be Imported, page C-44. For more information, see Show Tables to be Imported, page C-44.

Importing Data from Specific Fields

You can customize which fields in the tables listed in CZ_XFR_TABLES are extracted and imported. See the Oracle Integration Repository for more information about CZ_XFR_TABLES and other control tables.

There is no concurrent program to complete this customization. Modification of specific fields can only be accomplished by using SQL.

Populating Import Tables

The import tables below are listed in the order in which the concurrent programs and SQL*Plus import procedures populate them. This order must not be modified.

- CZ_IMP_ITEM_TYPE
- CZ_IMP_PROPERTY
- CZ_IMP_ITEM_TYPE_PROPERTY
- CZ_IMP_ITEM_MASTER
- CZ_IMP_ITEM_PROPERTY_VALUE
- CZ_IMP_DEVL_PROJECT
- CZ_IMP_LOCALIZED_TEXTS
- CZ_IMP_PS_NODES

Modifying EXPLOSION_TYPE

You can modify the CZ_XFR_PROJECT_BILLS.EXPLOSION_TYPE field for previously imported bills to indicate how the BOM Model exploder should handle standard Items. The possible values for this field are OPTIONAL (default), ALL, or INCLUDED. The EXPLOSION_TYPE refers to whether the component is mandatory (ALL or INCLUDED) or optional (OPTIONAL). See the Oracle Integration Repository for more information about CZ_XFR_PROJECT_BILLS and other control tables.

Identifying a BOM Model for Import

CZ_XFR_PROJECT_BILLS.TOP_ITEM_ID is the Oracle Inventory identifier of the BOM Model imported into the CZ schema. Every imported BOM Model must be represented in CZ_XFR_PROJECT_BILLS.

The TOP_ITEM_ID and ORGANIZATION_ID for each imported BOM Model are read from the CZ_XFR_PROJECT_BILLS table. The PS_NODE import updates the CZ_XFR_PROJECT_BILLS table with the timestamp, ID, and description of the most recent import.

The ORGANIZATION_ID also identifies which BOM Models are imported. Oracle Configurator uses the ORGANIZATION_ID when adding a configured line Item in Order Management. An order line is only valid if it contains the ORGANIZATION_ID that corresponds to the ORGANIZATION_ID on BOM Model Items in Oracle Applications.

For detailed information about the control tables, see the Oracle Integration Repository.

Importing Decimal or Integer Quantities

During import, CZ_PS_NODES.DECIMAL_QTY_FLAG is set to 1 if all of the following conditions are true:

- The BOM Model component is a Standard Item (CZ_IMP_PS_NODES.BOM_ITEM_TYPE=4 or CZ_PS_NODES.PS_NODE_TYPE=438)
- The corresponding Oracle Inventory Item has MTL_SYSTEM_ITEMS.INDIVISIBLE_FLAG='N' or 'NULL'
- The Model containing the Standard Item is an ATO Model (that is, CZ_DEVL_PROJECTS.MODEL_TYPE='A')
- The profile option CZ: Populate Decimal Quantity Flags is set to 1 (Yes)

CZ_PS_NODES. DECIMAL_QTY_FLAG is set to false if the imported Model Item is an Option Class, the Standard Item's parent is not an ATO Model, or the CZ: Populate Decimal Quantity Flags is set to No. Only Standard Items within ATO BOM Models support decimal quantities. Models, Option Classes and Standard Items within PTO

BOM Models do not support decimal quantities.

You can specify whether Items are imported as integers or decimals using the profile option CZ: Populate Decimal Quantity Flags. The CZ: Populate Decimal Quantity Flags profile option specifies whether and how the MTL_SYSTEM_ITEMS.INDIVISIBLE_FLAG for an Item should determine the value of the DECIMAL_QTY_FLAG column in both CZ_ITEM_MASTERS and CZ_PS_NODES.

- If the profile option is set to `No`, then import populates the DECIMAL_QTY_FLAG column in both CZ_ITEM_MASTERS and CZ_PS_NODES with a value of 0.

- If the profile option is set to `Yes`, then the value of MTL_SYSTEM_ITEMS.INDIVISIBLE_FLAG for an Item determines the value of the DECIMAL_QTY_FLAG column in both CZ_ITEM_MASTERS and CZ_PS_NODES.

    - If INDIVISIBLE_FLAG is 0 or NULL, then DECIMAL_QTY_FLAG in both tables is set to 1, which means that decimal quantities are allowed.

    - If INDIVISIBLE_FLAG is 1, then DECIMAL_QTY_FLAG in both tables is set to 0, which means that decimal quantities are not allowed. The minimum, maximum, and quantity are rounded during import. If the result of the rounding causes the minimum to be greater than the default or the maximum, then an error is returned.

    - If INDIVISIBLE_FLAG is 0 and a node cannot support decimal quantities based on the new restrictions, then any decimal values that occur in a BOM Model are rounded. This includes child Models and Option Classes within PTO Models.

If you change the profile option from `No` to `Yes`, then you must refresh all existing Models so they reflect the decimal quantity setting for each Oracle Inventory Item. You must also republish any existing publications.

For general information about using CZ: Populate Decimal Quantity Flags, see the *Oracle Configurator Installation Guide*.

> **Warning:** Not all Oracle Applications that are integrated with Oracle Configurator support decimal quantities for BOM Model Standard Items. Additionally, Oracle Configurator offers limited support for using decimal quantities. See specific product documentation in Applications Documentation, on the Oracle Technology Network to find out whether an application supports decimal quantities.

See the *Oracle Configurator Developer User's Guide* for additional information on the impact of decimal quantities on configuration models and rules. For information about how decimal quantities affect the CIO, see the *Oracle Configurator Extensions and Interface Object Developer's Guide.*

Importing Minimum and Maximum Instances

The first time a BOM Model is imported, the minimum and maximum Instance setting is 1. Subsequently, the BOM Model's minimum and maximum Instance may be changed in Oracle Configurator Developer, but refreshing the BOM Model does not override the minimum and maximum Instance values. The minimum and maximum Instance settings can only be set on a referenced BOM Model, never on the root Model. Refreshing the BOM Model does update the Quantity. See Refreshing Imported Data, page 5-16 for more information on refreshing Model data.

Importing the Data

Data can be imported into the CZ schema by:

- Running the Populate and Refresh Configuration Models Concurrent Programs, page C-18 in Oracle Applications. These concurrent programs import BOM Model structure (ATO, PTO Models, structure and rules) and require that the BOM Models be complete and identified at the specified root. For more information, see Populate and Refresh Configuration Models Concurrent Programs, page C-18.

- Running the Import Configuration Rules, page C-23 concurrent program in Oracle Applications. This concurrent program imports rules written in CDL format into the CZ schema. For more information about rule import, see Rule Import, page 5-21.

- Customizing your data import to run or suppress the transfer of some data. For more information, see Controlling the Data for Import, page 5-11.

- Running the PL/SQL IMPORT_SINGLE_BILL, page 18-26 procedure. For more information, see in Procedures and Functions in the CZ_modelOperations_pub Package, page 18-11.

- Running the PL/SQL REFRESH_SINGLE_MODEL, page 18-32 procedure. For more information, see Procedures and Functions in the CZ_modelOperations_pub Package., page 18-11

If you are not importing from the same remote (import) server from which you originally imported the BOM Models, then you must synchronize your BOM-based configuration models with the BOM Models on the new import server. For more information, see Synchronizing Data, page 7-1.

Imported BOM Models are read-only in Oracle Configurator Developer, although you can add Properties, create additional Model structure, and define rules when defining your BOM-based configuration model.

See Importing a BOM Model That Contains Other BOM Models, page 5-17 and the *Oracle Configurator Developer User's Guide* for the specific results in Oracle Configurator Developer when importing BOM Models.

Verifying the Data Import

After you import data into the CZ schema, view the Item Master and updated Model(s) in Oracle Configurator Developer. All Items imported into the CZ schema are displayed in the Oracle Configurator Developer Item Master. All imported CDL rules are displayed in either the Model's Configuration Rules folder or the folder that you specify in CZ_IMP_RULES.FOLDER_ID. All imported rules appear as Statement Rules. Imported BOM rules as mentioned in Inventory and BOM Data That Can Be Imported, page 5-6 do not appear in the Model's Configuration Rules folder. For more information on importing rules, see Rule Import, page 5-21.

The status of the import can be determined by examining the DISPOSITION field in the CZ_IMP tables. For more information about the DISPOSITION field see Import Control Fields, page 4-4.

Refreshing Imported Data

When changes are made in a production instance, it is necessary that the Models in the development instance be refreshed so that they reflect the changes. Refreshing configuration models only refreshes the data on the development CZ schema (target database instance).

Oracle Configurator's Refresh All Imported Configuration Models, page C-22 concurrent program updates all configuration models in the development CZ schema with changes that have been made in the production CZ schema. When you refresh BOM Models that have submodels, all changes that were made in the BOM Model and its submodels are reflected in Oracle Configurator Developer.

The refresh concurrent programs ensure that existing production data, such as saved configuration data, is preserved. The procedures that perform the refresh prevent customer-specific groups of fields in the CZ schema from being altered or nulled out even when other fields in the row are replaced during a refresh request. After the Refresh All Imported Configuration Models, page C-22 or Refresh a Single Configuration Model, page C-21 concurrent program is run, the Models must be republished to the production CZ schema. See Publishing Configuration Models, page 16-1 and the *Oracle Configurator Developer User's Guide* for additional publishing information.

> **Warning:** If you are using a separate development database, then you must never Generate Logic, Refresh or Create a User Interface, or run any schema maintenance scripts against a production database. Never use Oracle Configurator Developer for any development work on a production database.

Refreshing Imported Data Recommendations

Oracle recommends that you limit changes to the source data during construction of a

configuration model to avoid potential problems introduced by interim data imports and updates. Oracle suggests that unit testing be completed before you import changes from Oracle Applications or legacy data, so that the test cases are up-to-date with the application that has been constructed. Your Model's full system testing should include importing changed data and upgrading Oracle Configurator to match current enterprise or legacy data before deploying the runtime Oracle Configurator. Test cases may have to be updated to match the changes.

Although randomly updating imported data in the CZ schema during a development phase is not recommended, Oracle recognizes that project managers may need to synchronize with Oracle Applications data frequently. Refreshes and updates require careful control of what data gets imported. Likewise, corrections to the definitions of the configuration model in the runtime Oracle Configurator should be carefully controlled. A refresh may cause deletion of previously imported data. For example, if components are deleted from a BOM Model, they are also deleted from the configuration model during the next refresh. If components are added to the BOM Model, they are added to the configuration model during the next refresh. Oracle Configurator's Disable/Enable Refresh of a Configuration Model, page C-23 concurrent program can be used to reduce the number of Models affected by a refresh by disabling or enabling specific configuration models. Oracle Configurator's Refresh a Single Configuration Model, page C-21 concurrent program, updates the single imported BOM Model data in the CZ schema with changes that may have been made in the BOM Model.

Refreshing Procedures

If you are refreshing configuration models based on BOM Models that were previously imported from Oracle Bills of Material, you must:

1. Ensure that the refresh of the configuration model is enabled (see Disable/Enable Refresh of a Configuration Model, page C-23)

2. Explode the BOM Models you want to import if you are not importing from the local server (see Exploding BOM Models in Oracle Applications, page 5-10)

3. Run the appropriate refresh concurrent program (see Refresh a Single Configuration Model, page C-21 or Refresh All Imported Configuration Models, page C-22)

After you refresh a BOM Model, all changes that were made in Oracle Bills of Material are reflected in Oracle Configurator Developer. For more information see the *Oracle Configurator Developer User's Guide*.

Importing a BOM Model that Contains Other BOM Models

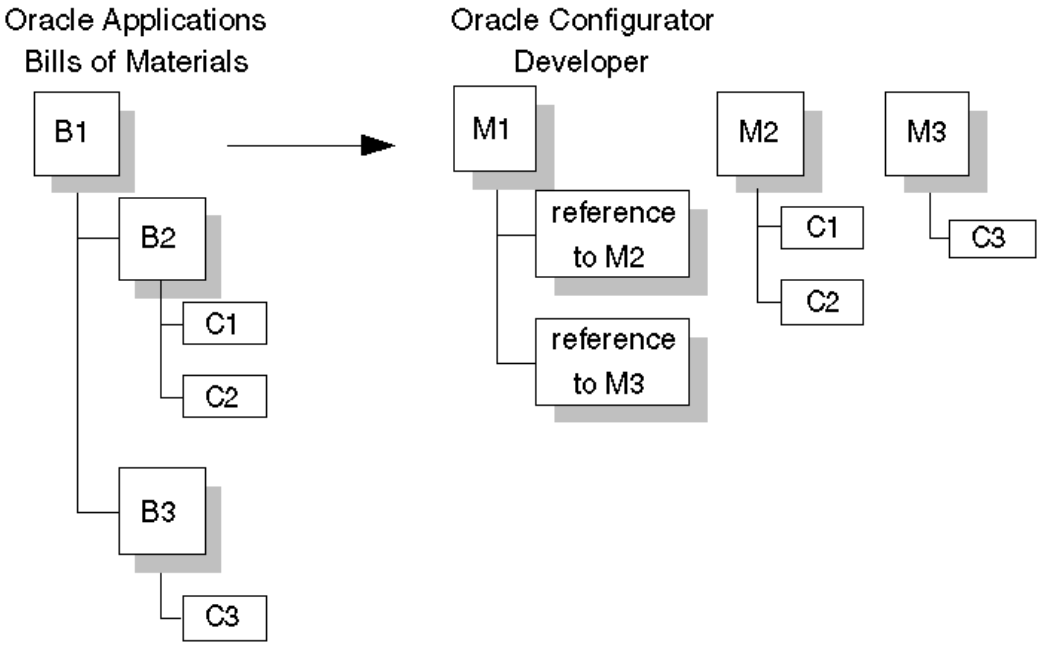
This section describes what exists in the CZ schema and is visible in Configurator Developer when you first import a BOM Model that contains other BOM Models from Oracle Bills of Material.

Example: Importing a BOM Model that Contains Other BOM Models

A BOM Model (B1) contains two child BOM Models (B2 and B3). Importing B1 results in three corresponding Models (M1, M2, and M3) in the CZ schema. All of these Models are visible in the Main area of the Configurator Developer Repository. Because B2 and B3 have child components in Oracle Bills of Material, M2 and M3 have corresponding children in Configurator Developer. See Initial Import of BOM Model with Submodels, page 5-18.

The Initial Import of BOM Model with Submodels, page 5-18diagram shows BOM Model B1 with children BOM Models B2 and B3. BOM Model B2 has children C1 and C2. BOM Model B3 has child C3. In Oracle Configurator Developer, Model M1 has References to Model M2 and Model M3. In Configurator Developer, Model M2 has two children, named C1 and C2. Model M3 has one child named C3.

Initial Import of BOM Model with Submodels



Refreshing a BOM Model that Contains Other BOM Models

This section explains what happens in Configurator Developer when you refresh a BOM Model in which the following changes have been made in Oracle Bills of Material:

- BOM Model References Have Changed, page 5-19
- BOM Models Referenced by Previously Imported BOM Model Have Changed, page 5-19

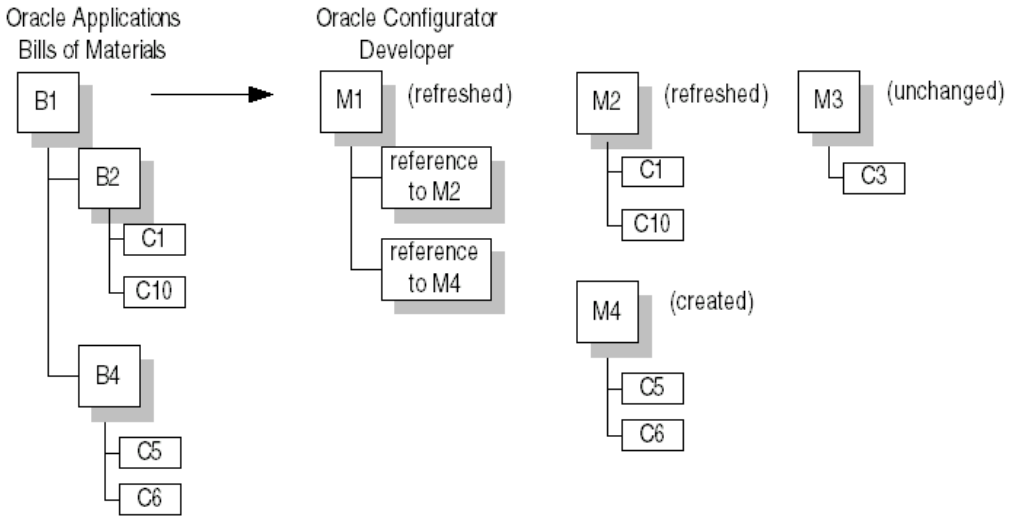
BOM Model References Have Changed

Replacing one child BOM Model for another in a BOM Model causes the root Model to be refreshed as expected. However, the child Model that was previously referenced is no longer referenced, but remains in the Configurator Developer Repository.

BOM Model B1 no longer references BOM Model B3, but now references BOM Model B2 and a new BOM Model B4. B2 has been modified to contain C1 and C10 and no longer contains C2. The new BOM Model B4 contains C5 and C6. When you populate or refresh BOM Model B1 by running either the Populate Configuration Models, page C-19 or Refresh a Single Configuration Model, page C-21 concurrent program, the corresponding Models M1 and M2 are refreshed in Oracle Configurator Developer. Model M4 is created to correspond to BOM Model B4 and Model M3 remains unchanged. Populate and Refresh Modified BOM Model , page 5-19 illustrates this result in Oracle Configurator Developer.

The Populate and Refresh Modified BOM Model , page 5-19 diagram shows BOM Model B1 with children BOM Models B2 and B4. BOM Model B2 has child nodes C1 and C10. BOM Model B4 has child nodes C5 and C6. A refresh of the BOM Model B1 is reflected in Oracle Configurator Developer with Model M1 referencing Models M2 and M4. Model M2 is refreshed with child nodes C1 and C10. Model M3 is unchanged with child node C3. Model M4 is created with child nodes C5 and C6.

Populate and Refresh Modified BOM Model



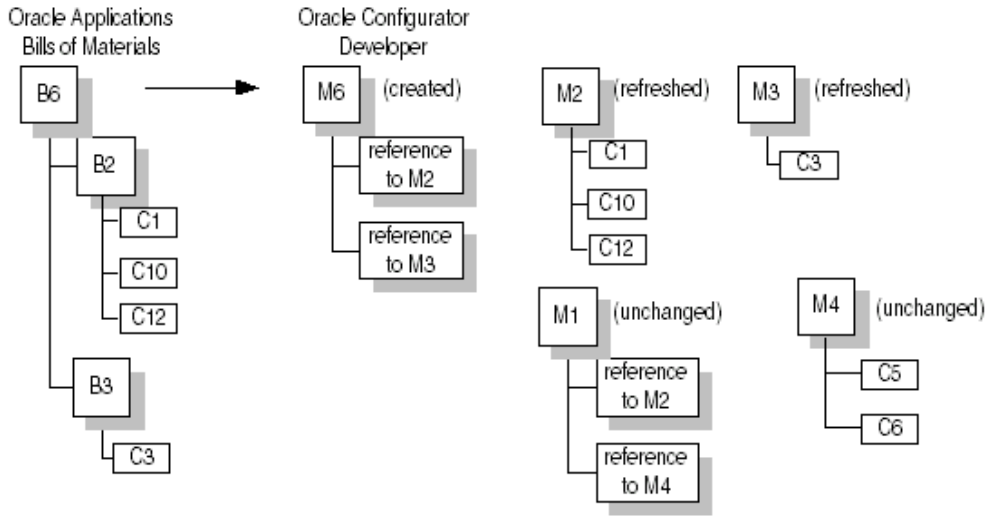
BOM Models Referenced by Previously Imported BOM Model Have Changed

Modifying and refreshing a child BOM Model that is referenced by numerous parent Models in Oracle Configurator Developer may cause the logic and UI of those parent Models to become invalid.

Using the example presented in Import a New BOM Model with References to Existing BOM Models, page 5-20, you create BOM Model B6 in Oracle Bills of Material. BOM Model B6 references BOM Models B2 and B3. When you import BOM Model B6 by running the Populate Configuration Models, page C-19 concurrent program, a new corresponding Model M6 appears in Oracle Configurator Developer as well as updated versions of Models M2 and M3. Model M1 now references the updated Model M2.

The Import a New BOM Model with References to Existing BOM Models, page 5-20 diagram shows BOM Model B6 with two child BOM Models B2 and B3. BOM Model B2 now has 3 child nodes C1, C10, and C12. BOM Model B3 has one child node C3. In Oracle Configurator Developer, Model M6 has references to Models M2 and M3. Oracle Configurator Developer has Model M2 refreshed with child nodes C1, C10, and C12. Model M3 is refreshed with child node C3. Model M1 is unchanged with references to the new refreshed Model M2 and Model M4 is unchanged in Oracle Configurator Developer.

Import a New BOM Model with References to Existing BOM Models



Models M1 and M6 both reference Model M2. When BOM Model B6 is imported into the CZ Schema, Model M2 is refreshed with a new child node C12. Model M1 is not refreshed. Importing Model M6 might create problems for Model M1 because the logic and UI may no longer be valid with the changes and updates. In this case, you must regenerate both the logic and the UI for Model M1.

If Model M1 was published before Model M2 was refreshed, then the runtime Oracle Configurator end user can still use Model M1 that references the original Model M2, as well as the publication of Model M6 that references the refreshed Model M2. This scenario is possible because the publishing process creates a copy of the configuration model at the time of publication.

For more information on publishing, see Publishing Configuration Models, page 16-1 and the *Oracle Configurator Developer User's Guide*.

BOM Model with a Common Bill

When a BOM Model that references a common bill is imported into the CZ schema, the imported BOM Model is available in the Main area of the Repository, but the common bill is not. When the imported BOM Model is opened in Configurator Developer, the components of the common bill appear as if the BOM was created with those components. The common bill is only available to the organization that imported the BOM Model. But when a common bill is imported directly (not as a reference), then the common bill is available to all organizations.

When you open the imported BOM Model for editing in the Structure area of the Workbench, the common bill's components are visible and available, but there are no visual clues indicating that the components are from a common bill.

When a BOM Model with references to BOMs is imported, the import procedure warns that a referenced BOM is being imported. When a BOM Model with references to a common bill is imported, there is no warning that the referenced bill is a common bill. For general information about common bills, see the *Oracle Bills of Material User's Guide*.

Rule Import

Configuration rules from legacy applications can be imported into the CZ schema. Before these rules can be imported into the CZ schema, they must be written in Constraint Definition Language (CDL) format. For information about writing rules in CDL format, see the *Oracle Configurator Constraint Definition Language Guide*. Rule Import Procedure, page 5-29 identifies the necessary tasks for importing these rules.

All rules imported in CDL format appear as Statement Rules in Oracle Configurator Developer. For more information about Statement Rules, see the *Oracle Configurator Developer User's Guide*.

Most types of rules can be written in CDL and imported into the CZ schema as Statement Rules:

- Logic rules
- Numeric contribution and consumption rules
- Comparison rules
- Property-based Compatibility rules

You *cannot* write the following types of rules in CDL, and consequently you cannot import them into the CZ schema as Statement Rules:

- Explicit Compatibility rules
- Design Charts

Note: Rules *cannot* be imported from a remote database. The source and target tables must be in the same database instance.

Related Topics

Rule Import Procedure, page 5-29

Populating CZ_IMP_RULES

The following fields must be populated in the CZ_IMP_RULES table before you can run the Import Configuration Rules, page C-23 concurrent program.

- ORIG_SYS_REF: A user-defined character string that identifies the rule as an imported rule.
- NAME: The name of the rule with a maximum of 255 characters
- RULE_FOLDER_ID: A number that identifies where the rule information is stored in CZ_RULE_FOLDERS. If this field is null, then the rule is stored in the Model's Configuration Rules folder.

  Once a rule is imported into the Model's Configuration Rules folder, you can move the rule to another rule folder associated with the Model.

  Note: If you move a rule to another rule folder, then you must specify the RULE_FOLDER_ID when you refresh the rule. If you do not specify the RULE_FOLDER_ID, then the refreshed rule will be moved into the Model's Configuration Rules root folder.

- DEVL_PROJECT_ID: The numeric identifier of the Model that is associated with the rule. This is a foreign key into CZ_DEVL_PROJECTS. DEVL_PROJECT_ID and must be the same number as CZ_IMP_LOCALIZED_TEXTS.MODEL_ID.
- RULE_TEXT: The actual CDL rule text
- RULE_TYPE: The numeric identifier of the type of rule. The imported rule is a Statement Rule and the RULE_TYPE is 200.

You should not populate the following fields in the CZ_IMP_RULES table:

- AMOUNT_ID
- ANTECEDENT_ID
- CHECKOUT_USER

- CLASS_NAME
- COMPONENT_ID
- CONSEQUENT_ID
- CREATED_BY
- CREATION_DATE
- DISPOSITION - See Import Configuration Rules, page C-23 for additional information
- EFF_FROM
- EFF_MASK
- EFF_TO
- EXPR_RULE_TYPE
- FSK_COMPONENT_ID
- FSK_DEVL_PROJECT
- FSK_LOCALIZED_TEXT_2
- FSK_MODEL_REF_EXPL_ID
- GRID_ID
- IMPORT_PROG_VERSION
- INSTANTIATION_SCOPE
- INVALID_FLAG
- LAST_UPDATED_BY
- LAST_UPDATE_DATE
- LAST_UPDATE_LOGIN
- MESSAGE
- MODEL_REF_EXPL_ID
- MUTABLE_FLAG

- PERSISTENT_RULE_ID
- PRESENTATION_FLAG
- REASON_ID
- REC_STATUS - See Import Configuration Rules, page C-23 for additional information.
- RULE_FOLDER_TYPE
- RULE_ID
- SEEDED_FLAG
- SEQ_NBR
- SIGNATURE_ID
- SUB_CONS_ID
- TEMPLATE_PRIMATIVE_FLAG
- TEMPLATE_TOKEN
- UI_DEF_ID
- UI_PAGE_ID
- UI_PAGE_ELEMENT_ID
- UNSATISFIED_MSG_ID

For more information about the CZ_IMP_RULES table, see the Oracle Integration Repository.

Related Topics

Rule Import Procedure, page 5-29

Populating CZ_IMP_LOCALIZED_TEXTS

Multiple Language Support data for rule violations and unsatisfied messages are stored in the CZ_IMP_LOCALIZED_TEXTS table. A single rule may have several records in the CZ_IMP_LOCALIZED_TEXTS table. If a rule has multiple translations, then there must be a record in CZ_IMP_LOCALIZED_TEXTS for each translation. All translation records for a single rule must have the same ORIG_SYS_REF.

For information on Multiple Language Support, see Multiple Language Support, page

14-1, the *Oracle Configurator Installation Guide*, *Oracle E-Business Suite Installation Guide: Using Rapid Install*, and *Oracle E-Business Suite Concepts*.

After you have created your CDL rule, you must populate the following fields in CZ_IMP_LOCALIZED_TEXTS table before running the Import Configuration Rules, page C-23 concurrent program.

- ORIG_SYS_REF: A user-defined character string that identifies the rule as an imported rule.
- LANGUAGE: The language code that is associated with the rule.
- SOURCE_LANG: The language code of the LOCALIZED_STR field.
- MODEL_ID: The DEVL_PROJECT_ID of the Model associated with the rule. The MODEl_ID must be the same number as CZ_IMP_RULES.DEVL_PROJECT_ID.
- LOCALIZED_STR: The rule's translated text.

You should not populate the following fields in the CZ_IMP_LOCALIZED_TEXTS table:

- CHECKOUT_USER
- CREATED_BY
- CREATION_DATE
- DISPOSITION - See Import Configuration Rules, page C-23 for additional information.
- EFF_FROM
- EFF_MASK
- EFF_TO
- INTL_TEXT_ID
- LAST_UPDATED_BY
- LAST_UPDATE_DATE
- LAST_UPDATE_LOGIN
- LOCALE_ID
- MESSAGE

- SEEDED_FLAG
- REC_STATUS - See Import Configuration Rules, page C-23 for additional information.
- FSK_DEVL_PROJECT_1_1
- IMPORT_PROG_VERSION

For more information about the CZ_IMP_LOCALIZED_TEXTS and CZ_INTL_TEXTS tables, see the Oracle Integration Repository.

Related Topics

Rule Import Procedure, page 5-29

Rule Import Tables

Every imported rule in CZ_IMP_RULES has a corresponding record in the CZ_RULE_FOLDER. The imported rule is linked to the specified Model's (DEVL_PROJECT_ID) Configuration Rules folder.

Tables for Importing Rules, page 5-27 describes the CZ tables that are used when importing rules.

Tables for Importing Rules

Table Name	Description
CZ_IMP_RULES	The source rule's data that is imported into the CZ_RULES in the CZ schema. The following columns are used when importing rules do not appear in the CZ schema: • MESSAGE - Is the error message if a rule is rejected during import. The rejection of a rule does not terminate the rule import request. A rejected rule is imported into the CZ schema. • RUN_ID - Is theParameter for the Import Configuration Rules Concurrent Program, page C-24. It is a generated number when the RUN_ID is not specified. • DISPOSITION - Is the result of processing the rule in the stage specified in REC_STATUS. For more information, see Stages of Rule Import , page 5-28. • REC_STATUS - Is the stage that the rule has been processed. For more information, see Stages of Rule Import , page 5-28 . • IMPORT_PROG_VERSION - Is the version of the import program that is used for importing data. The default value is 1.0.

Table Name	Description
CZ_IMP_LOCALIZED_TEXTS	The rule's translation data that is imported into the CZ schema. The following columns are used when importing rules and do not appear in the CZ schema: • MESSAGE - Is the error message if a rule is rejected during import. The rejection of a rule does not terminate the rule import request. A rejected rule is imported into the CZ schema. • RUN_ID - Is theParameter for the Import Configuration Rules Concurrent Program, page C-24. It is a generated number when the RUN_ID is not specified. • DISPOSITION - Is the result of processing the rule in the stage specified in REC_STATUS. For more information, see Stages of Rule Import , page 5-28. • REC_STATUS - Is the stage that the rule has been processed. For more information, see Stages of Rule Import , page 5-28 . • IMPORT_PROG_VERSION - Is the version of the import program that is used for importing data. The default value is 1.0.

Stages of Rule Import

Each rule goes through three processing stages before it is imported into the CZ schema. The rule's processing stage is tracked in CZ_IMP_RULES.REC_STATUS and CZ_IMP_LOCALIZED_TEXTS.REC_STATUS. The result of each processing stage is tracked in CZ_IMP_RULES.DISPOSITION and CZ_IMP_LOCALIZED_TEXTS.DISPOSITION. For more information about REC_STATUS and DISPOSITION during rule import, see Import Control Fields, page 4-4.

After all rules have been processed, the rules that have REC_STATUS=XFR and DISPOSITION = I or M are parsed.

Related Topics

Rule Import Procedure, page 5-29

Rule Validation

During rule import, the following fields are checked. If the field meets the criteria stated below, then an error message is stored in CZ_IMP_LOCALIZED_TEXTS.MESSAGE.

- CZ_IMP_LOCALIZED_TEXTS.ORIG_SYS_REF is null or belongs to a different Model
- CZ_IMP_LOCALIZED_TEXTS.LANGUAGE is null
- CZ_IMP_LOCALIZED_TEXTS.MODEL_ID - is null or refers to an invalid Model.
- CZ_IMP_LOCALIZED_TEXTS.SOURCE_LANG is null
- CZ_IMP_RULES.ORIG_SYS_REF is null
- CZ_IMP_RULES.NAME is null
- CZ_IMP_RULES.MODEL_ID is null or refers to an invalid Model

Rule Import Procedure

Importing rules into the CZ schema consists of the following steps:

1. Write the rule in CDL format.
2. Verify that the Model associated with the rule exists in the CZ schema. Note the Model's DEVL_PROJECT_ID. The DEVL_PROJECT_ID is used when you populate the CZ_IMP_LOCALIZED_TEXTS and CZ_IMP_RULES tables.
3. Populate the CZ_IMP_RULES table. See Populating CZ_IMP_RULES, page 5-22 for a list of fields that must be populated for each rule.
4. Populate the CZ_IMP_LOCALIZED_TEXTS table. See Populating CZ_IMP_LOCALIZED_TEXTS , page 5-24 for a list of fields that must be populated for each rule.
5. Run the Import Configuration Rules, page C-23 concurrent program.

   The Import Configuration Rules, page C-23 concurrent program validates the rules and stores the CDL format in the Rules subschema. Rule Validation, page 5-29 lists the fields that are examined when validating a rule during rule import.

   For more information about the concurrent program, see Import Configuration Rules, page C-23.
6. Edit the rules that had parsing errors as reported in the concurrent program log file.

All rules processed by the Import Configuration Rules, page C-23 concurrent program are imported into the CZ schema regardless of whether they have parsing errors. Once the rules are in the CZ schema, they can be edited in Configurator Developer or in the legacy environment and then refreshed.

Warning: If a rule is edited in both the legacy environment and the Configurator Developer environment and you refresh the rule, then the refreshed rule overwrites any changes that may have been made to the rule in the Developer environment.

Custom Import

A custom import is required for importing data not handled by a standard import, including legacy data from non-Oracle Applications databases. See Standard Import, page 5-4 to determine whether your data requires a custom data import. This section describes:

- Overview of Custom Data Import, page 5-30
- Identifying Data for a Custom Data Import , page 5-31
- Custom Import Procedure, page 5-34
- Required ASCII File Format for Custom Import, page 5-32
- Loading Property Values by Type, page 5-33

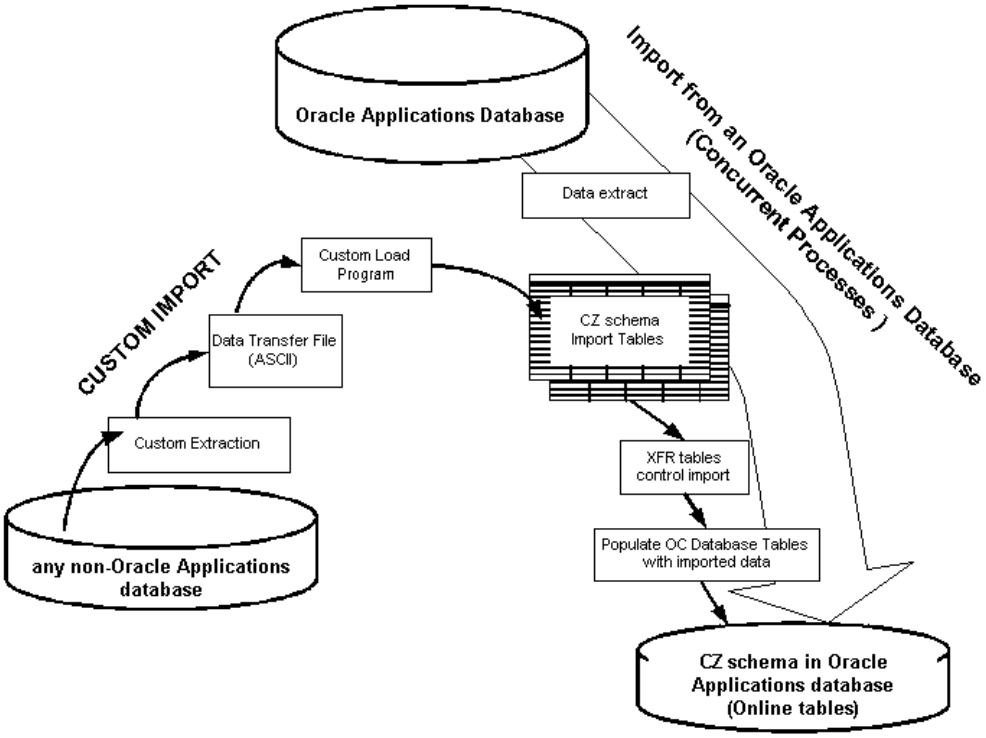
Overview of Custom Data Import

Both the standard and custom data import processes use the import tables in the CZ schema to populate the online tables. However, while data extraction for a standard import is handled by the Populate and Refresh Configuration Models Concurrent Programs, page C-18, a custom import requires custom extraction, transfer, and load into the import tables. Comparison of Custom and Standard Data Import, page 5-31 shows where in the process the two kinds of data import are different.

The Comparison of Custom and Standard Data Import, page 5-31 shows that when doing a custom import, custom load programs load the data into the CZ schema import tables and then the data is imported into the CZ schema online tables. Importing from Oracle Applications the data from Inventory and BOM schemas is copied to the CZ import and online schemas.

This figure also shows the comparison of the custom import processes and the processes when importing a BOM Model from the Oracle Applications.

Comparison of Custom and Standard Data Import



When importing data not handled by a standard import, especially non-Oracle legacy data, the data must be custom loaded into the import tables. Custom programs then populate the online tables with the extracted data. The data that is imported depends on the settings in the control tables (CZ_XFR_ tables in the CZ schema) and the custom load program, if applicable. See Custom Import Procedure, page 5-34 for information about performing a custom import.

After successfully importing any legacy data needed for modeling new configurations, Oracle recommends that you unit test your configuration model before transferring new or updated model data. Unit testing configuration models is performed in the Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for more information.

Identifying Data for a Custom Data Import

The following tables can be populated through a custom import:

- CZ_DEVL_PROJECTS
- CZ_INTL_TEXTS
- CZ_ITEM_MASTERS

- CZ_ITEM_PROPERTY_VALUES
- CZ_ITEM_TYPES
- CZ_ITEM_TYPE_PROPERTIES
- CZ_LOCALIZED_TEXTS
- CZ_PROPERTIES
- CZ_PS_NODES

Minimally, the following tables are used for custom import and should be selected when you run the Select Tables To Be Imported concurrent program:

- CZ_ITEM_MASTERS
- CZ_ITEM_TYPES
- CZ_ITEM_TYPE_PROPERTIES
- CZ_ITEM_PROPERTY_VALUES
- CZ_PROPERTIES

To know what data to extract for populating the import tables, you need to know what fields are available in the import tables for data population. See the Oracle Integration Repository, for detailed information about all import table fields. See also Dependencies Among CZ Schema Import Tables, page 4-8, for information about the dependencies among the import tables.

As with a standard data import, you can further control the data populating the online tables by using the control tables (CZ_XFR_). See Controlling the Data for Import, page 5-11 for details.

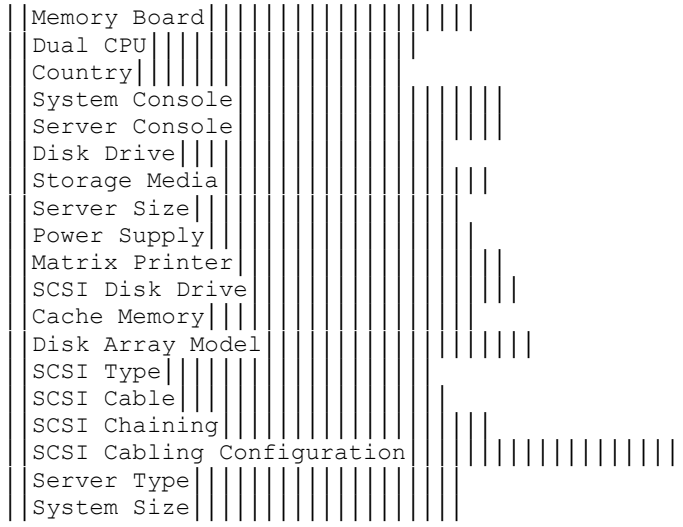
Custom import programs should consider the setting of QUOTEABLE_FLAG in the CZ_PS_NODES table. This flag determines whether or not the OC Servlet's UI Server displays a particular Item in the Configuration Summary page. For more information about the Summary page see the *Oracle Configurator Developer User's Guide*.

Required ASCII File Format for Custom Import

The format of the data transfer files must exactly match the target import tables, field for field. The data transfer files include all data in text (ASCII) format, with fields separated by delimiters such as a vertical bar (|).

Data Transfer File Format, page 5-33 shows a data transfer file that imports Item types.

Data Transfer File Format



Loading Property Values by Type

When preparing source data for custom import, your custom load programs should place Property data values into the import tables according to their data type. The CZ_IMP tables provide separate columns for numeric and non-numeric Property values, and for default values.

The table Columns for Imported Property Values, page 5-33 shows which column to load Property values into, depending on the data type of the value.

Columns for Imported Property Values

Data Type	If CZ_IMP_PROPERTY.DATA_TYPE is...	Load default Property value into CZ_IMP_PROPERTY column ...	Load value on the Item into CZ_IMP_ITEM_PROPERTY_VALUE column ...
integer	1	DEF_NUM_VALUE	PROPERTY_NUM_VALUE
decimal	2	DEF_NUM_VALUE	PROPERTY_NUM_VALUE
Boolean (values are 0 or 1)	3	DEF_NUM_VALUE	PROPERTY_NUM_VALUE
text	4	DEF_VALUE	PROPERTY_VALUE

Custom Import Procedure

To import data that is not handled by a standard import:

1. Identify and cleanse data for import.

2. Create and run custom extraction programs for the data you want to import.

   Creating a custom extraction program

   1. Write queries to extract the data into the required data transfer file format required by the import tables.

   2. Optionally create an ASCII file in that data transfer (DAT) format (see Required ASCII File Format for Custom Import, page 5-32).

   3. Write a load program that loads the data transfer file into the import tables, or loads the queried data directly into the import tables in the format required.

3. Optionally set up the CZ control tables to customize the transfer of data (see Controlling the Data for Import, page 5-11).

4. Run the cz_modeloperations_pub.import_generic PL/SQL procedure. For more information see IMPORT_GENERIC, page 18-27.

5. Verify your import as described in Verifying the Data Import, page 5-16.

6

Migrating Data

This chapter describes how to migrate a CZ Release 12 instance into an empty CZ instance, and how to migrate Model data from one instance to another development instance.

This chapter covers the following topics:

- Introduction
- Migrating Data from a CZ Schema
- Migrating Models

Introduction

Migrating data is the process of copying data from one database instance to another database instance. You can migrate data from a CZ schema to another, empty CZ schema, or migrate Models from one development instance to another. The latter process is explained in Migrating Models, page 6-3.

The process of migrating data from a CZ schema should only be run against a target database containing a new installation of Oracle Applications, and both the source target and database instances must have the same schema version.

Migrating data from a CZ schema does not:

- Transfer data from the CZ_IMP_ tables
- Transfer data from custom tables that are not in the CZ schema
- Transfer saved configurations

  Migrating saved configurations is not recommended because it typically involves a very large amount of data.

Warning: Data migration is a one-time process. Once migration is

complete, do not repeat the process or use the migration concurrent programs to refresh data in the Oracle Applications database.

Migrating Data from a CZ Schema

To migrate an Oracle Configurator Release 12 schema:

1. Check the versions of the Oracle Configurator Release 12 source and target database schemas.

   Both the source and the target must be at the same minor version. If there is a difference between the two database schema versions, then migration cannot continue. You must take appropriate steps, such as upgrading, to bring either the source database instance or the target database instance to the desired version.

   See Verifying CZ Schema Version, page B-3 for details.

2. Verify that there are no implementors logged in to Configurator Developer that is connected to the either the migration source or target database instances.

3. Verify that there are no end users connected to either the migration source or target database instances, including production deployments or a test runtime Oracle Configurator.

4. Delete Models from the Oracle Configurator Developer Repository that do not need to be migrated into the target database schema.

5. Run the Purge Configurator Tables, page C-4 concurrent programs to clean up the source schema prior to migrating the data. For more information see Purge Configurator Tables, page C-4.

6. Verify that the target CZ schema is empty before you run the Setup Configurator Data Migration, page C-30 concurrent program on the target database instance.

   Note: An "empty" schema means a new Oracle Applications installation that contains no data. If you must remove data from an existing instance and use it as a target of data migration, then contact Oracle Support Services for assistance.

7. Run the Migrate Configurator Data, page C-32 concurrent program from the target database instance. For more information, including possible issues recorded in the log file, see Migrate Configurator Data, page C-32.

8. Resolve all issues or errors that are reported in the log file.

9. Verify that the Import Enabled flag on the source database instance is enabled. For more information, see Enable Remote Server, page C-12.

10. If you will be importing BOM Model data to the target instance from a different server from which you migrated the data, then run the Synchronize All Models, page C-27 concurrent program on the target instance.

    For example, your import source is DB-X and you import BOM Model data from DB-X to DB-1. You then migrate data from DB-1 to DB-2. If you will be importing data to DB-2 from any database other than DB-X, then you must run the Synchronize All Models program on DB-2.

    For more information on BOM Model synchronization, see The BOM Model Synchronization Process, page 7-2.

Migrating Models

Migrating models is the process of copying configuration model data from one instance (the source) into another development instance (the target). Imported BOM Models and Models created in Configurator Developer can be migrated, and you can migrate one or multiple Models at the same time. After the migration process completes successfully, you can view and modify the migrated Models in Oracle Configurator Developer on the target instance.

> **Note:** Do not confuse Model migration with data migration. Data migration refers to migrating the *entire* CZ schema into an empty database instance, whereas Model migration refers to migrating specific Models and their structure, rules, and UIs to another database instance.

When you migrate a Model for the first time, it creates a Model with the same name on the target instance. If you migrate the same Model again, a new copy is created on the target, and its default name is "*Model Name* (Migrated from *source DB name: Repository Folder path*." Any subsequent migration of the Model creates Models called "*Model Name* (Migrated from *source DB name: Repository Folder path* copy *x*)." Migrating a Model creates a copy of that Model on the target instance, regardless of whether the Model has changed on the source instance. For details about how the migration process handles referenced Models, see Migrating Referenced Models, page 6-5.

To migrate a Model, the target database instance must be a development instance (it cannot be a publication target), and both the source and target instances must have the same schema version.

The following objects that are part of or are associated with the selected Model are copied to the target instance:

- Model structure

- Rules
- User Interfaces
- Referenced and connected Models
- UI Templates
- Usages
- Effectivity Sets
- Configurator Extension Archives
- Populators and Item Master data
- Properties

For details about data synchronization criteria and a list of what is not copied during migration, see Synchronizing Migrated Model Data, page 6-8.

Migrating Models is a two step process:

1. In Configurator Developer, specify which Models you want to migrate.

   For details, see the *Oracle Configurator Developer User's Guide*.

2. Run the Migrate Models concurrent program.

   See Migrate Models, page C-38.

These steps can be performed only by the Oracle Configurator Administrator.

> **Note:** A Model's compiled logic is not migrated to the target instance. After migrating a Model, you must open the Model for editing in Configurator Developer on the target instance and generate logic. Additionally, you may need to update saved configurations of the migrated Model so they can be restored successfully. For details, see Restoring Saved Configurations of Migrated Models, page 6-7.

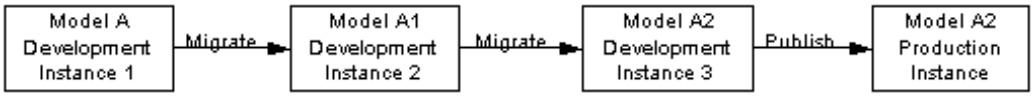
Model Migration Examples

The following examples show possible scenarios in which it may be useful or necessary to migrate Models. It should be noted however that these examples are not intended to show best practices or a recommended way of building and maintaining configuration models.

The illustration Migrating Models Serially, page 6-5 shows the migration of a Model across several development instances before being migrated to the production instance. In this scenario, each development instance is used to develop a specific area of the Model, such as Model structure in Instance 1, rules in Instance 2, and UIs in Instance 3.

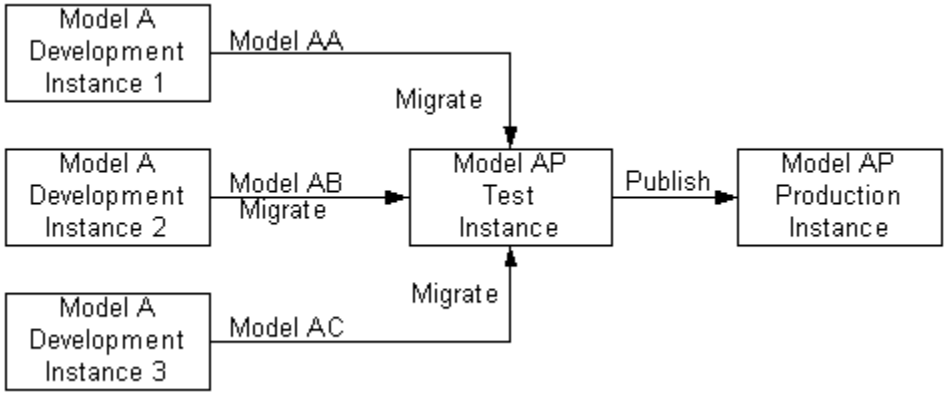
After unit and system testing is complete, the final version of the Model (A2) in Instance 3 is published to the Production instance.

Migrating Models Serially



The illustration Migrating Models from Multiple Development Instances, page 6-5 shows the migration of a Model from several development instances to a single test instance, before publishing to a production instance. In this scenario, several different areas of a Model are developed in isolation in separate development instances and then the Models are migrated and synchronized in the test instance before being published to the production instance.

Migrating Models from Multiple Development Instances



A configuration model that has been published can be migrated into another development instance for additional modification without affecting the published Model.

Migrating Referenced Models

The Migrate Models concurrent program detects whether the root Model (parent) or one of its referenced (child) Models has been previously migrated to the target instance, and whether the child Model has changed on either the source or target since the last migration.

If you migrate a Model that references a previously migrated Model, and the child Model has not changed on the source instance, then the migrated parent refers to the existing child on the target instance. If the child Model has changed on either the source or the target instance since the last migration, then a new copy of that Model is created on the target instance. If the child Model does not yet exist on the target, then it is

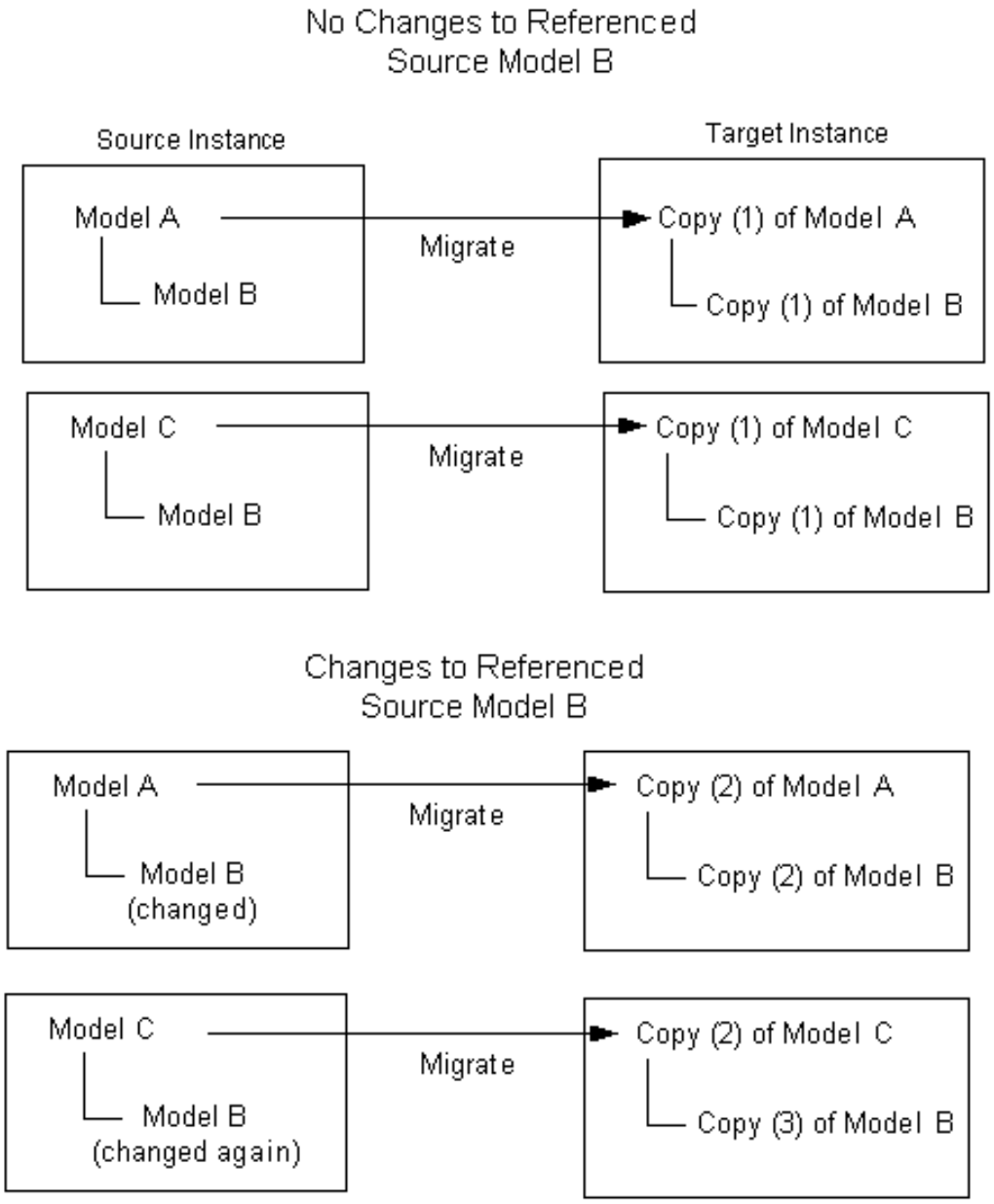
migrated (for example, a new reference was created in the parent Model on the source instance).

The illustration Migration of Referenced Models, page 6-7 shows examples of Models with referenced Models that are migrated to another instance. When Model A is migrated, copies of both Model A and its child Model B are created on the target instance. You then migrate Model C, which also references Model B. Model B has not changed since the last migration, so the copy of Model C references the existing copy of Model B on the target.

Model B is then modified on the source instance (for example, new Model structure is added), so when you remigrate Model A, a new copy of both Model A and its child Model B are created. Model B is modified again on the source, so when you remigrate Model C, new copies of both Model C and its child Model B are created on the target.

The migration process does *not* create a new copy of a referenced Model if only the Model's name has changed.

Migration of Referenced Models



Restoring Saved Configurations of Migrated Models

It is possible for the process of migrating Models to create a situation in which the persistent node identifier (persistent_node_id) that Oracle Configurator uses internally to identify configuration nodes are not able to uniquely identify Model nodes when restoring a saved configuration. This can occur when, for example, a pre-Release 12 instance that contains saved configurations is upgraded to Release 12 or later. In Release

12 and later, saved configurations store both node names and a persistent node ID.

The following concurrent programs add Model node names to saved configurations and enable Oracle Configurator restore them successfully:

- Add Model Node Names to Configurations by Model Items, page C-35
- Add Model Node Names to Configurations by Product Key, page C-37

You must run one of these concurrent programs on a migration target instance if all of the following are true:

- The instance was upgraded to Release 12 (or later) from a prior release
- You intend to publish Models from the instance
- You want to be able to restore configurations of published migrated Models on the instance

Synchronizing Migrated Model Data

When synchronizing Model data, the Migrate Models concurrent program matches the source configuration model's internal keys that refer to BOM and Inventory data with comparable data on the target instance. The data are comparable when the Item and Inventory Organization names of the data being migrated match data of the same type on the target instance. All of the migrated BOM Model's referenced Models are also synchronized with comparable BOM Models on the target instance. For more information, see Migrating Referenced Models, page 6-5.

After the synchronization completes successfully, the Item IDs for the migrated BOM Model reflect the IDs on the target instance. If the name or structure of the BOM Items on the source and target instances is different, then the migration fails. For more information, see Result of Synchronizing BOM Models, page 7-6.

During Model migration, the following occurs:

- Populators that reference Item Master data such as Item IDs, Property IDs, and/or Item-Type IDs, are changed to reference the migrated data. However, Populators that match Item Master data by name are not changed during migration.
- Item Master data that exists in the source but not in the target is migrated. This includes missing items, values, and assignments. Existing Item data on the target instance is not changed. The migration process never overwrites or modifies any existing data on the target instance.
- Repository objects such as Usages, Properties, and Effectivity Sets that exist in the source but not in the target are migrated.

  Because multiple Models may refer to the same Repository object, creating multiple copies of that object on the target instance is both unnecessary and undesirable.

Therefore, synchronization that occurs during Model migration redirects any references to Repository objects on the source instance to equivalent Repository objects on the target instance.

The log file that is created when you run the Migrate Models concurrent program describes the results of the migration. An example of this file is shown below.

Example of a Log File Generated by the Migrate Models Concurrent Program

```
Item 'SMX_1 Model' already exists on the migration target instance.  All
migration objects in the migration target instance that reference 'SMX_1
Model', will be changed to use the Item.
.
.
.
The Effectivity Set for rule 'LR-O11 R O21' already exists in the
migration target instance. Rule 'LR-O11 R O21' is part of the Rule
Sequence 'RS'. All rules in Rule Sequence 'RS' will be changed to 'Never
Effective'.
.
.
.
Effectivity Set 'Eff - Delete 1' already exists on the migration target
instance.  All migration objects in the migration target instance that
reference 'Eff - Delete 1', will be changed to use the Effectivity Set.
```

To view this log file, click View Log from the Requests page after the Migrate Models concurrent program completes successfully.

For more information, see Synchronization Criteria During Model Migration, page 6-9
.

Synchronization Criteria During Model Migration

The following table provides the synchronization criteria for the different types of Configurator data that is not Model-specific. The columns include Object Type, Matching Criteria, Synchronization Condition, and Result.

Object Type is the data that is migrated and used to match the comparable Model's object. Matching Criteria is the specified Object type's data that is used for matching. Synchronization Condition is the condition that must exist for the object to pass synchronization. The Result column indicates the end result of the migration based on the synchronization condition.

Object Type	Matching Criteria	Synchronization Condition	Result
Property	Property Name, src_application_id	Data Type and default value are the same	Migrated Model is changed to reflect existing data on the target instance

Object Type	Matching Criteria	Synchronization Condition	Result
		Data Type is the same, Property default value is different	Default Property value on the migrated Model will be different than on the source Model
		Data Type is different	Migration fails
Item Type	Item Type Name, src_application_id	Item Type Name and Properties are the same	Migrated Model is changed to reflect existing Item Type on the target instance
		Additional Properties exist on the Item Type in the target instance	Migrated Model acquires any additional Properties on nodes that are associated with that Item Type
		Source Item Type has additional properties	Migration fails
Item	Item name (ref_part_nbr), src_application_id	Item Type Name and Item Property values are the same	Migrated Model is changed to refer to existing Item on the target instance
		Item Type is the same, but Item Property values are different	Migrated Model is changed to refer to existing Item Type on the target instance, property values remain unchanged on target
		Item Type does not match	Migration fails

Object Type	Matching Criteria	Synchronization Condition	Result
UI Content Template	UI_DEF_ID = 0, TEMPLATE USAGE = '0', Template Name	TEMPLATE TYPE, MESSAGE_TYPE, ROOT_ELEMENT_TYPE, MAIN_MESSAGE_ ID, PARENT CONTAINER TYPE, ROOT ELEMENT SIGNATURE ID, and ROOT_REGION_TYPE are the same	Migrated Model is changed to refer to existing UI Content Template on the target instance
		TEMPLATE TYPE, MESSAGE_TYPE are the same, but ROOT_ELEMENT_TYPE, MAIN_ MESSAGE_ ID, PARENT CONTAINER TYPE, ROOT ELEMENT SIGNATURE ID, or ROOT_REGION_ TYPE are different	Migration fails
		Mismatch on TEMPLATE TYPE or MESSAGE_TYPE	The template is migrated to the target instance, it is renamed (if a Template with the same name exists on the target), Template references are changed to reflect the change, and a warning message is displayed.
UI Master Template	Name, UI_DEF_ID = 0	(no conditions)	Migrated Model UIs always acquire the target's Master Template's characteristics

Object Type	Matching Criteria	Synchronization Condition	Result
Effectivity Set	Name	(no conditions)	The Migrated Model acquires the settings of the matching Effectivity Set on the target (see Migrating Effectivity Sets Used in Rule Sequences, page 6-13))
Usage	Name	Usage Name is the same	Update to indicate the ID of the Usage on the target
		Name does not match and there are less than 64 active Usages on the target	Usage is migrated
		Name does not match and there are 64 active Usages on the target	Migration fails
Archive (for Configurator Extensions)	Archive_name	(no conditions)	Migrated Model uses the target's existing Archive
Populator	Item ID, Property ID, Item Type ID	(no conditions)	If the Populator references Item Master data by ID (Item ID, Property ID, or Item Type ID) then it is changed to reference the data on the target. If the Populators match Item Master data by name, then the Populator is not changed.

Object Type	Matching Criteria	Synchronization Condition	Result
BOM Import Source	Import server on source and target instances	The source BOM Model's import source does not match the target BOM Model's import source	The BOM Model's import source is changed to reflect the import source for the target

Migrating Effectivity Sets Used in Rule Sequences

When you migrate a Model, all of the Model's Rule Sequences are also migrated to the target instance. If any of the rules in a Rule Sequence refer to an Effectivity Set that exists on the target instance, all of the rules in the Rule Sequence will be set to 'Never Effective' after the migration is complete. (This change will be listed in the Migrate Models concurrent program log file. For details, see Synchronizing Migrated Model Data, page 6-8). You may want to modify the Rule Sequence on the target instance and specify new effectivity dates for each rule in the set.

If a rule refers to an Effectivity Set that exists on the target, and the rule is *not* part of a Rule Sequence, then the rule refers to the Effectivity Set on the target after the model is migrated.

7

Synchronizing Data

This chapter describes when and how data is synchronized. This includes synchronizing BOM data after the import server has changed and synchronizing publication data after a database has been cloned.

This chapter covers the following topics:

Overview

This chapter describes when and how data is synchronized. This includes synchronizing BOM data after the import server has changed and synchronizing publication data after a database has been cloned. This chapter explains how to restore the identity and linkage of mismatched data by:

- Synchronizing BOM Model Data, page 7-2
- Synchronizing Publication Data, page 7-6

Introduction

The kinds of data and circumstances requiring synchronization are:

- BOM Models
    - The import server has changed to a different database instance (for example, you previously imported BOM data from instance A, but you now import BOM data from instance B)

    - The production database instance is not the import server
    - The import source or import target data has been migrated to another database instance
- Configuration model publication records
    - The Publication source or target database instance has been cloned
    - Publication data has been migrated to another database instance

Publication synchronization must be run after BOM Model synchronization only when data is migrated from one database instance to another. In all other scenarios, the two kinds of synchronization are independent from one another. For more information on migration, see Migrating Data, page 6-1.

For information about synchronizing BOM Model data, see Synchronizing BOM Model Data, page 7-2.

For information about synchronizing publication records on cloned database instances, see Synchronizing Publication Data after a Database Instance is Cloned, page 7-7.

Synchronizing BOM Model Data

The configuration model in the CZ schema is an extension of the source BOM Model that participates in Oracle Applications processes such as ordering. For a BOM Model to be orderable, the BOM Model in the CZ schema must match certain criteria with the BOM Model in Oracle Bills of Material. Synchronization causes the BOM-based configuration model in the CZ schema to be modified to match the production BOM Model.

Data synchronization is not the same as data refresh (see Refreshing Imported Data, page 5-16).

The concurrent programs for synchronizing BOM Model data are described in Model Synchronization Concurrent Programs, page C-25.

The BOM Model Synchronization Process

The process for synchronizing BOM Model data is as follows:

1. Check the similarity between the production BOM Model you wish to use as the new import source or publication target, and the BOM Model represented in your configuration model.

   For more information, see Checking BOM and Model Similarity, page 7-3.

2. Synchronize the BOM Model in the configuration model with the source BOM Model by running the Synchronize All Models concurrent program. For more

information, see Result of Synchronizing BOM Models, page 7-6.

3. After synchronizing the BOM-based configuration model with the source BOM Model, you can proceed with any of the following:

   - Reimport or refresh the BOM Model in the CZ schema (see Populating the CZ Schema, page 5-1)

   - Publish the configuration model (see Publishing Configuration Models, page 16-1)

   Running the publication concurrent programs includes BOM Model synchronization. For details, see Publishing a Configuration Model, page 16-10.

Checking BOM and Model Similarity

The two concurrent programs available for checking if the BOM Model in the CZ schema sufficiently matches the source BOM Model are:

- Check Model/Bill Similarity

- Check All Models/Bills Similarity

For details about these concurrent programs, see Check Model/Bill Similarity, page C-25 and Check All Models/Bills Similarity, page C-27.

Running the Check Model/Bill Similarity and Check All Models/Bills Similarity concurrent programs creates a Check Model/Bill Similarity report, which describes the fields that do not match and must be corrected before synchronization can occur. For more information, see Criteria for BOM Model Similarity, page 7-3. For more information about the report, see Model/Bill Similarity Check Report, page C-28.

Criteria for BOM Model Similarity

The Check Model/Bill Similarity, page C-25 and Check All Models/Bills Similarity, page C-27 concurrent programs use validation criteria to determine if a BOM-based configuration model is similar enough to be synchronized with the source BOM Model:

- Both structures use the same Inventory Items. For example: The bill's Item identity is identified by the concatenated values of segments 1 through 20 in MTL_SYSTEM_ITEMS of the corresponding Item. CZ_PS_NODES are identified by the corresponding value of CZ_ITEM_MASTERS.REF_PART_NBR.

- Parent-child relationships are the same in the source and target BOM Models. For example, each imported parent node has the same imported children Items as in the BOM Model structure. The order of the children may be different.

- Certain Item characteristics are the same. For example, the value of minimum or

maximum default quantities, or the 'Required when parent is selected' Property are the same.

- A child's effectivity range does not fall outside the effectivity range of its parent.
  - If there is only one child node with the given identity (CONCATENATED_SEGMENTS), then its disable date (effective to date) should be the same as the parent node and the effective dates (effective from date) should either be before SYSDATE or be the same for the child node and the parent.
  - If there is more than one child node with the given identity (CONCATENATED_SEGMENTS), then the previous scenario is only valid for the child node that has the earliest effective date. For the other child nodes the ranges should be exactly the same.
- When creating a BOM Model through an interface, records may not be recognized by Oracle Configurator during the synchronization process if the BOM_INVENTORY_COMPONENTS.IMPLEMENTATION_DATE field is null. If this field is null, then it is automatically populated with either the EFFECTIVITY_DATE or the SYSDATE.

Fields That Must Be Synchronized , page 7-4 lists the configuration model's data fields that must be synchronized with the import source BOM Model or publication target.

The following table lists the appropriate table for synchronization and the fields that are synchronized for import.

Fields That Must Be Synchronized

Table	Field	Import	Publication
CZ_DEVL_PROJECTS	ORIG_SYS_REF includes back pointers to EXPLOSION_TYPE:ORGANIZATION_ID:TOP_ITEM_ID	Yes	Yes
CZ_ITEM_MASTERS	ORIG_SYS_REF includes back pointers to INVENTORY_ITEM_ID:ORGANIZATION_ID	Yes	Yes
CZ_ITEM_TYPES	ORIG_SYS_REF includes back pointers to ITEM_CATALOG_GROUP_ID	Yes	Yes

Table	Field	Import	Publication
CZ_LOCALIZED_TEXTS	ORIG_SYS_REF includes back pointers to COMPONENT_ITEM_ID:EXPLOSION_TYPE:ORGANIZATION_ID	Yes	No
CZ_MODEL_PUBLICATIONS	PRODUCT_KEY includes back pointers to ORGANIZATION_ID:TOP_ITEM_ID	Yes	Yes
	ORGANIZATION_ID	Yes	Yes
	TOP_ITEM_ID	Yes	Yes
CZ_PS_NODES	ORIG_SYS_REF includes back pointers to COMPONENT_CODE:EXPLOSION_TYPE:ORGANIZATION_ID:TOP_ITEM_ID	Yes	Yes
	COMPONENT_SEQUENCE_PATH	Yes	Yes
	COMPONENT_SEQUENCE_ID	Yes	Yes
CZ_XFR_PROJECT_BILLS	ORGANIZATION_ID	Yes	No
	TOP_ITEM_ID	Yes	No
	COMPONENT_ITEM_ID	Yes	No
	SOURCE_SERVER	Yes	No

Organization information is mapped by matching ORG_ORGANIZATION_DEFINITIONS.ORGANIZATION_CODE. If the matching Organization is not found, then an error occurs.

> **Note:** It is important that the Item flexfield structure and the concatenation characters for the Item flexfield be the same on all database instances and not updated.

BOM Model synchronization checks the Models that are candidates for synchronization but results in an error if a Model does not have an EXPLOSION_TYPE of OPTIONAL. See Modifying EXPLOSION_TYPE, page 5-13 for more information about the EXPLOSION_TYPE setting. BOM Model synchronization does not check the mandatory fields.

Result of Synchronizing BOM Models

After determining that the source BOM Model and the BOM-based configuration model are sufficiently similar, based on the report generated by the Check Model/Bill Similarity, page C-25 and Check All Models/Bills Similarity, page C-27 concurrent programs, the BOM Models can be synchronized either by running the Synchronize All Models, page C-27 or the publication concurrent programs.

Attempting to synchronize mismatched BOM Models results in errors.

BOM synchronization causes the Item identification in the BOM-based configuration model to be matched with the import source or publication target BOM Model. During data import, the CZ schema is populated with the source BOM Model's ORIG_SYS_REF identification. However, the same BOM Model in Bills of Material of two different database instances may have different ORIG_SYS_REF identification.

If the database instance from which the BOM Model was imported into the CZ schema is replaced with a new instance containing the same BOM Model, it is likely that the ORIG_SYS_REF identification longer will no longer match the original source BOM Model. Likewise, if the configuration model is being published to an instance that did not serve as the import server, the ORIG_SYS_REF identification may not match the source BOM Model.

Because CZ_ITEM_TYPE_PROPERTIES and CZ_ITEM_PROPERTY_VALUES do not have the ORIG_SYS_REF field, there is no way for the Check Model/Bill Similarity, page C-25 and Check All Models/Bills Similarity, page C-27 concurrent programs to verify that the imported Properties and Property values correspond to the Descriptive Elements and their values on the target instance. Runtime Models use the imported Property values. You must manually verify that the Descriptive Elements and their values are the same on both the source and target of the BOM Model synchronization.

Synchronizing Publication Data

Publication data can become inconsistent when you

- Clone a publication source or target database instance
- Migrate data from one database instance to another
- Decommission the production or target database instance

After changing databases in these ways, you must synchronize the publication data so that inconsistencies are corrected. Examples of data inconsistencies are:

- Missing publications
- Incorrect publications
- Overlapping publications
- Missing or incorrect entries in the CZ_SERVERS table

The concurrent programs for synchronizing publication data are described in Publication Synchronization Concurrent Programs, page C-40.

See Publishing Configuration Models, page 16-1 for details about creating publications, and about the relationship between the publication data on the source and target database instances.

Synchronizing Publication Data after a Database Instance is Cloned

Cloning can be done into a new empty database instance or into one that already contains work product data. In either case, the cloned database contains a copy of the original data, but publication data becomes inconsistent in the following ways.

- References between the source and target publications can become lost or incorrect
- Applicability parameters of publication records on the source and target can overlap

Publication data inconsistencies need to be resolved by updating data on both the cloned and the publication source or on the target that was not cloned. The following publication synchronization concurrent programs are available after cloning either a target or source database instance:

- Synchronize Cloned Target Data, page C-40 synchronizes the publication data in the new cloned target database with the publication data on the source database.
- Synchronize Cloned Source Data, page C-41 synchronizes the publication data in the new cloned source database with the publication data on the target database.

See Example of Synchronizing Publication Data on a Cloned Target, page 7-9 for details about the circumstances and results of synchronizing a cloned publication target. See Example of Synchronizing Publication Data on a Cloned Source , page 7-12for details about the circumstances and results of synchronizing a cloned publication source.

> **Warning:** After cloning a publication source, do not clone the target until you have first synchronized publications on that cloned source, or vice versa.

Example of Synchronizing Publication Data

The example illustrating publication synchronization uses CZ_SERVERS and CZ_MODEL_PUBLICATIONS data to illustrate where inconsistencies occur between a publication source and target after cloning or restoring a source or target database instance from backup.

CZ_SERVERS Table

Publication synchronization updates the CZ_SERVERS table to ensure that the local and remote servers are listed correctly to associate the cloned publication source or target with the appropriate publication records on the unchanged target or source, respectively.

CZ_MODEL_PUBLICATIONS Table

The following columns in the CZ_MODEL_PUBLICATIONS table help identify target publications relative to their source so they can be republished:

- PUBLICATION_ID, page 7-8
- REMOTE_PUBLICATION_ID, page 7-8
- SERVER_ID, page 7-8

PUBLICATION_ID

PUBLICATION_ID is the publication's generated identifier in the database instance containing the configuration model. This identifier is generated when a publication record is created in the Create Publication page.

REMOTE_PUBLICATION_ID

REMOTE_PUBLICATION_ID on the source database instance points to the PUBLICATION_ID on the target database instance. The REMOTE_PUBLICATION_ID on the target database instance points to the PUBLICATION_ID on the source database instance. See Original Publication, page 7-9.

SERVER_ID

SERVER_ID associates the publication record with a database instance in the CZ_SERVERS table.

Example Publication Data Before Cloning

The following examples of publication data presume a publication source database, A, with PUBLICATION_ID 1000 and a publication target database, B, with PUBLICATION_ID 2000. Original Publication, page 7-9 shows the original publication records on Source A and Target B.

In the publication record on Source A:

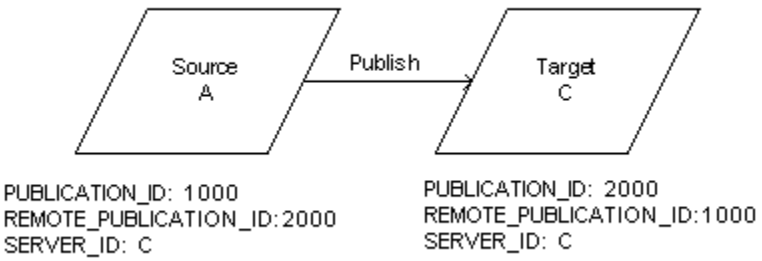
- REMOTE_PUBLICATION_ID is 2000 because it points to the PUBLICATION_ID on the publication target
- SERVER_ID of the publication record is B because it points to the LOCAL SERVER_ID on the publication target

In the publication record on Target B:

- REMOTE_PUBLICATION_ID is 1000 because it points back to the PUBLICATION_ID on the publication source
- SERVER_ID of the target publication record is B, because it identifies itself as the LOCAL entry in the CZ_SERVERS table

Original Publication, page 7-9 illustrates a publication record on the source and target databases. Source A's REMOTE_PUBLICATION_ID 2000 references Target B's PUBLICATION_ID 2000, and Target B's REMOTE_PUBLICATION 1000 references the Source A's PUBLICATION_ID 1000. Source A's server ID points to the Target B's server. Target B's server ID points to itself, not to Source A's server.

Original Publication



Publication records on the target assume only one publication source and do not identify the source publication record by the SERVER_ID of the source.

Example of Synchronizing Publication Data on a Cloned Target

Synchronizing publication data on a cloned target resolves the following issues caused by cloning the publication target:

- The CZ_SERVERS table on the source does not include a listing for the cloned target.
- A database link must be established between the publication source and the cloned target.

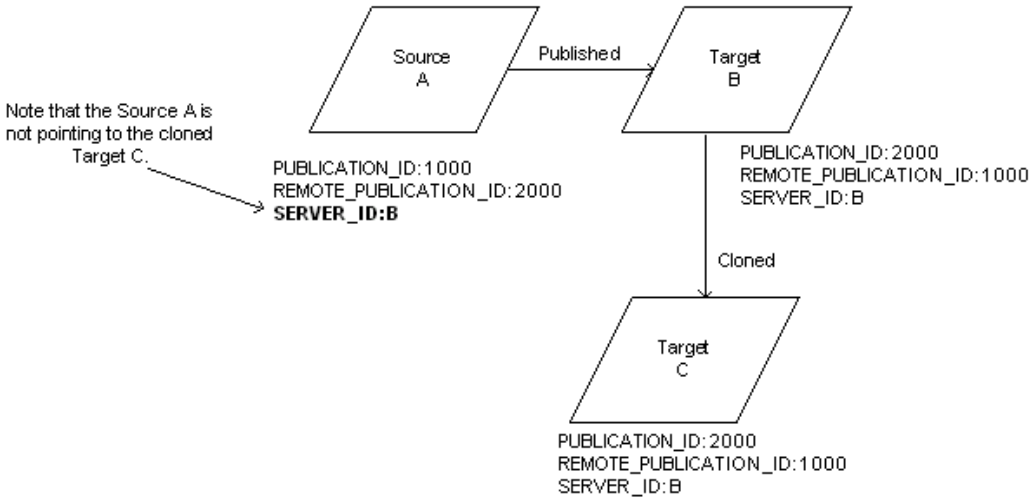
- References to the publication record on the source database instance are lost, wrong, or have overlapping applicability parameters.

Original Publication, page 7-9 shows the original publication records on Source A and Target B.

Target B is then cloned to create Target C. Publication After Cloning, page 7-10 illustrates the resulting cloned Target C copy. The publication record on Source A does not point to the cloned publication record on cloned Target C. Source A *still references* Target B as the target server for the publication record (SERVER_ID:B).

Publication After Cloning, page 7-10 illustrates a publication record after Target B is cloned to Target C. Target C's publication record has the same values as the original target publication record. The original Target B's REMOTE_PUBLICATION_ID 1000 refers to Source A's PUBLICATION_ID 1000. The cloned Target C's REMOTE_PUBLICATION_ID 1000 does not have any indication that this is referencing a record on Source A.

Publication After Cloning

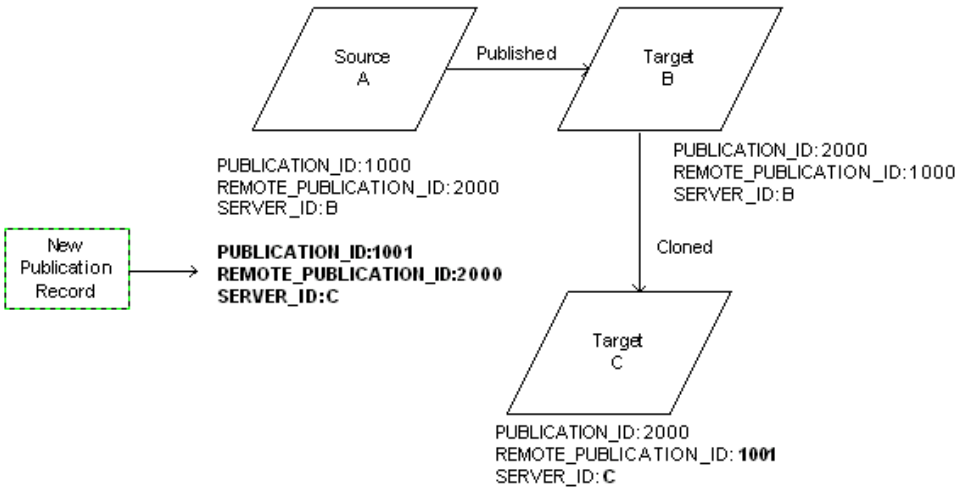


Source A is then synchronized with Target C. Publication After Synchronization, page 7-11 illustrates the resulting publication information after synchronization. A *new* publication record is created on Source A referencing the record on cloned Target C. The publication record on cloned Target C is also updated so that it references the new publication record on Source A as well as correcting the SERVER_ID that associates the publication record with a LOCAL database instance.

Publication After Synchronization, page 7-11 illustrates a publication record after synchronizing Source A and Target Target C. Cloned Target C's publication record is updated with a new REMOTE_PUBLICATION_ID that now references a new publication record created on Source A. The new publication record's REMOTE_PUBLICATION_ID on Source A references the updated publication record

on the cloned Target.

Publication After Synchronization



Example of Missing Source Publication, page 7-11 summarizes the publication information from the original publication to the cloning, to the synchronization.

The following table is a summary of what happens after cloning and then synchronizing a publication.

Example of Missing Source Publication

	Source A	Target B	Target C (cloned from B)
Original publication:			
PUBLICATION_ID	1000	2000	
REMOTE_PUBLICATION_ID	2000	1000	
SERVER_ID	B	B	
After Cloning Target B to Target C:			
PUBLICATION_ID	1000	2000	2000

	Source A	Target B	Target C (cloned from B)
REMOTE_PUBLICATION _ID	2000	1000	1000
SERVER_ID	B	B	B
After Synchronizing Source A and Target C:			
PUBLICATION_ID	1000	2000	2000
REMOTE_PUBLÌCATION _ID	2000	1000	updated
SERVER_ID	B	B	updated
PUBLICATION_ID	1001		2000
REMOTE_PUBLICATION _ID	2000		1001
SERVER_ID	C		C

For information on running the Synchronize Cloned Target Data, page C-40 concurrent program, see .

Example of Synchronizing Publication Data on a Cloned Source

Synchronizing publication data on a cloned source resolves the following issues caused by cloning the publication source:

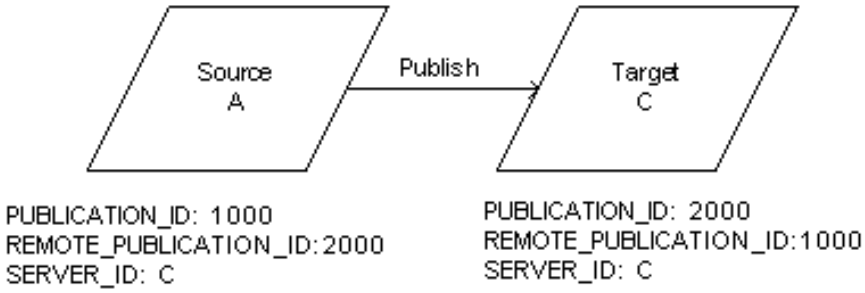
- The CZ_SERVERS table on the cloned source contains incorrect information in the LOCAL server entry of the clone.
- The SOURCE_SERVER_FLAG on the publications target identifies the original source, not the cloned source as the publication source server.
- A database link must be established between the publication target and the cloned source.
- Target publication records require only one corresponding publication source.

Note: Oracle does not support publishing from multiple source database instances to a single target database instance. Oracle recommends decommissioning original source when synchronizing the cloned source.

Publication Before Cloning the Source Database, page 7-13 illustrates a Model that is originally published from Source A to Target C.

Publication Before Cloning the Source Database, page 7-13 illustrates a publication record before Source A is cloned. Source A's REMOTE_PUBLICATION_ID 2000 references Target C's PUBLICATION_ID 2000, and Target C's REMOTE_PUBLICATION_ID 1000 references Source A's PUBLICATION_ID 1000.

Publication Before Cloning the Source Database



CZ_SERVERS Entries on Source A Before Cloning, page 7-13 illustrates some of the entries for database instances A and C in the CZ_SERVERS table of Source A before cloning.

The following table illustrates database instance entries on two servers prior to cloning the source server.

CZ_SERVERS Entries on Source A Before Cloning

Server	LOCAL_NAME	SERVER_LOCAL_ID	HOSTNAME	DB_LISTENER_PORT	INSTANCE_NAME
A	LOCAL	0	my_serv	1521	A
C	SALES	1	my_serv	1521	C

CZ_SERVERS Entries on Target C Before Cloning, page 7-14 illustrates some of the entries for database instances A and C in the CZ_SERVERS table of Target C before

cloning.

The following table illustrates database entries on two servers before cloning the target server.

CZ_SERVERS Entries on Target C Before Cloning

Server	LOCAL_NAME	SERVER_LOCAL_ID	HOSTNAME	DB_LISTENER_PORT	INSTANCE_NAME
A	source	1	my_serv	1521	A
C	LOCAL	0	my_serv	1521	C

The SOURCE_SERVER_FLAG on Target C is set to 1, meaning Target C recognizes Source A as its publication source.

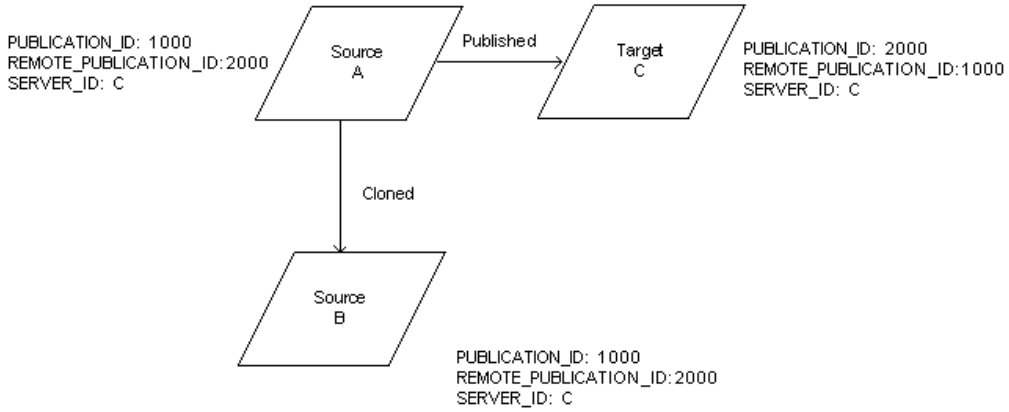
If configuration models are published from Source A to Target C, and then Source A is cloned to create Source B, the following inconsistencies occur:

- The LOCAL entry in the CZ_SERVERS table of Source B must be updated by removing the entry for Source A and completing the identification for Source B.
- The publication record on Source A and its clone on Source B both point to Target C which is incorrect.
- Publication records on Target C continue to identify Source A as the publication source server.

Source Server B is Cloned from Source Server A, page 7-15 illustrates Source B as a clone of Source A.

Source Server B is Cloned from Source Server A, page 7-15 illustrates a publication record after Source A is cloned to Source B. Source B's publication record has the same values as the original source publication record. Both the cloned and the original Source A's REMOTE_PUBLICATION_ID 2000 refers to Target C's PUBLICATION_ID 2000. There is no way for the publication on Target C to know that its source is now Source B.

Source Server B is Cloned from Source Server A



After cloning, the clone's CZ_SERVERS table is an exact copy of the original Source A (see CZ_SERVERS Entries on Source A Before Cloning, page 7-13). Source B must be synchronized because its CZ_SERVERS table does not have a LOCAL entry for Source B.

To synchronize existing publications records on Source B with Target C, and publish new Models from B to C, you must first run the Synchronize Cloned Source Data, page C-41 concurrent program on Source B.

Running the Synchronize Cloned Source Data, page C-41 concurrent program updates the LOCAL entry in the CZ_SERVERS table on Source B with correct information. CZ_SERVERS Entries on Server B After Synchronization, page 7-15 shows the entries on the two servers in the CZ_SERVERS table on B after running the synchronization concurrent program.

CZ_SERVERS Entries on Server B After Synchronization

Server	LOCAL_NAME	SERVER_LOCAL_ID	HOSTNAME	DB_LISTENER_PORT	INSTANCE_NAME
B	LOCAL	0	my_serv	1521	B
C	SALES	1	my_serv	1521	C

Synchronizing Source B has no effect on Target C. By publishing or republishing a Model from Source B to Target C, the CZ_SERVERS table on Target C is updated. CZ_SERVERS Entries on Target C After Publishing a Model from Source B, page 7-16 shows Source B listed as the publication source in the CZ_SERVERS table on Target C, with the SOURCE_SERVER_FLAG enabled (set to 1). Both Source A and Source B can serve as publication source.

The following table illustrates the target server settings after publishing a Model from the source server.

CZ_SERVERS Entries on Target C After Publishing a Model from Source B

Server	LOCAL_NAME	SERVER_LOCAL_ID	HOSTNAME	DB_LISTENER_PORT	INSTANCE_NAME	SOURCE_SERVER_FLAG
A	source	1	my_serv	1521	A	1
B	source	2	my_serv	1521	B	1
C	LOCAL	0	my_serv	1521	C	0

If a decision is made to *not* decommission Source A, and there are configuration models that were published from A to C, then running the Synchronize Cloned Source Data, page C-41 concurrent program on Source B removes any cloned publications to prevent conflict between the two publications sources and allows Source A to continue as the source for those publications.

> **Note: Republish** and **New Copy** in the Model Publications page are disabled for a disabled publication record. Oracle Configurator Developer users can delete the disabled publication record or edit the publication's applicability parameters to re-enable the publication in Production or Test mode.

8

CZ Schema Maintenance

This chapter explains how to maintain data when it exists in more than one place and is potentially unsynchronized.

This chapter covers the following topics:

- Overview
- Introduction
- Refreshing or Updating the Production CZ Schema
- Purging Configurator Tables
- Redoing Sequences

Overview

Data that is maintained in more than one place is subject to becoming out of synch. This chapter presents the following processes to help you keep multiple data sources synchronized:

- Refreshing or Updating the Production CZ Schema, page 8-2
- Purging Configurator Tables, page 8-2
- Redoing Sequences, page 8-4

Introduction

Inventory and Bills of Material data must be maintained in the production instance. You can maintain the CZ schema with the data in the production instance by:

- Refreshing or Updating the Production CZ Schema, page 8-2
- Eliminating any unused data by Purging Configurator Tables, page 8-2

- Redoing Sequences, page 8-4 resets the sequences after the CZ schema has been restored from a dump file
- Synchronizing BOM Model Data, page 7-2

Refreshing or Updating the Production CZ Schema

When a runtime Oracle Configurator is deployed, the data is stored in the CZ schema directly through networked use. During deployment, further imports are performed to refresh the CZ schema as Oracle Applications or legacy data changes. The procedures that perform the import prevent customer-specific groups of fields in the CZ schema from being altered or nulled out even when other fields in the row are replaced during an import session.

For additional information about refreshing data in your CZ schema, see Refreshing Imported Data, page 5-16.

Purging Configurator Tables

Large databases affect performance. For example, large amounts of data in the import tables may cause data import to fail. The following concurrent programs delete unnecessary data:

- Purge Configurator Tables, page 8-2
- Purge Configurator Import Tables, page 8-3
- Purge To Date Configurator Import Tables, page 8-3
- Purge To Run ID Configurator Import Tables, page 8-3

> **Note:** A data import session must not be running when there is a purge concurrent program request. Similarly, a purge session must not be running when there is a data import concurrent program request.

Purge Configurator Tables

The Purge Configurator Tables, page C-4 concurrent program physically deletes all logically-deleted records in the tables and subschemas of the CZ schema.

Each CZ schema table has delete-propagation rules that affect the results of running the Purge Configurator Tables concurrent program.

The Purge Configurator Tables concurrent program:

- Propagates deletions to additional records not marked as deleted, such as

physically deleting children of a logically-deleted PS_NODE record.

- Physically deletes all EXPRESSION_NODE records attached to a deleted rule.
- Does not physically delete a record that is logically-deleted if there is a non-deleted reference to that record, such as preserving a deleted PS_NODE that is used in a non-deleted rule.

See Purge Configurator Tables, page C-4 for details on running this concurrent program.

Purge Configurator Import Tables

Import performance can be improved if you purge the import tables in your database instance. The Purge Configurator Import Tables, page C-5 concurrent program deletes data in all CZ_IMP tables. The concurrent program also deletes the corresponding data in the CZ_XFR_RUN_INFOS and CZ_XFR_RUN_RESULTS control tables.

See Purge Configurator Import Tables, page C-5 for running this concurrent program.

Purge To Date Configurator Import Tables

If you want to improve import performance but also retain recent import information, then the Oracle Configurator Administrator should run the Purge To Date Configurator Import Tables, page C-6 concurrent program. Unlike the Purge Configurator Import Tables, page C-5 concurrent program that deletes all data in the CZ_IMP tables, the concurrent program only deletes the oldest data in the CZ_IMP tables. The data for the specified past number of days is retained. The concurrent program also deletes the corresponding data in CZ_XFR_RUN_INFOS, and CZ_XFR_RUN_RESULTS control tables.

See Purge To Date Configurator Import Tables, page C-6 for details on running this concurrent program.

Purge To Run ID Configurator Import Tables

If you want to improve import performance but also retain recent import run information, then the Oracle Configurator Administrator should run the Purge To Run ID Configurator Import Tables, page C-7 concurrent program. Purge To Run ID Configurator Import Tables, page C-7 only deletes data in the CZ_IMP tables up to the specified input Run ID. The concurrent program also deletes the corresponding data in the CZ_XFR_RUN_INFOS, and CZ_XFR_RUN_RESULTS control tables

See Purge To Run ID Configurator Import Tables, page C-7 for details on running this concurrent program.

Redoing Sequences

After restoring a schema from a backup file, you should refresh the database sequences. The REDO_SEQUENCES procedure is invoked by the packages CZ_MANAGER.sql and CZ_*subschema*_MGR.sql (for example, CZ_PS_MGR.sql).

Depending on the parameters that you enter, the REDO_SEQUENCES procedure either alters or recreates the sequence objects in the database that are used to allocate primary keys for tables in the CZ schema. The procedure checks the current high primary key value in the database and sets a new start value that is greater than the current high value. The procedure uses the default incremental value specified by OracleSequenceIncr setting in the CZ_DB_SETTINGS table unless you specify a new increment. See OracleSequenceIncr, page 4-19 for more information.

Part 3

Integration

Part 3 presents integration information for setting up Oracle Configurator with other Oracle Applications or a custom application as described in Integration Tasks, page 1-4.

9

Session Initialization

This chapter describes the format and parameters of the initialization message for the runtime Oracle Configurator.

This chapter covers the following topics:

- Overview
- Introduction
- Setting Parameters
- Initialization Parameter Types
- Initialization Parameter Descriptions

Overview

This chapter describes the format, parameters, and use of the initialization message for the runtime Oracle Configurator, including information about:

Note: If your host application is part of Oracle Applications, then the initialization message is already defined, and you do not need to define it yourself. However, this chapter may be of great value to you in understanding how that initialization message calls the runtime Oracle Configurator.

If your host application is a custom application, then you must define your own initialization message, as described in this chapter.

Introduction

See Configurator Architecture, page 2-1 for an explanation of the interaction between the elements discussed in this chapter.

In a typical host application (such as a web store), a button, tab, or similar control is coded so that it launches the runtime Oracle Configurator, allowing the end user to configure a model of a product or service. For the purposes of this explanation, think of this control as "the Configure button". This chapter describes how to make the Configure button select the wanted configuration model and user interface in the runtime Oracle Configurator.

Definition of Session Initialization

Session initialization takes place when your host application calls the runtime Oracle

Configurator and renders your configuration model in the user interface you have specified. The *initialization message* allows a host application to start a configuration session with specified characteristics.

When you set the parameters of the initialization message in your host application, your parameters handle the types of responsibilities listed in Initialization Parameter Types, page 9-9.

When your host application calls the runtime Oracle Configurator, the initialization message is sent to the Oracle Configurator Servlet, using the HTTP POST method. (POST is used in preference to GET to accommodate the length of the message).

See Invocation of Oracle Configurator by Host Application, page 2-4 for a description of how the initialization message is routed, depending on the requirements of the host application, and the type of user interface.

The initialization message is written in XML, and has `<initialize>` as its document element. You must specify the parameters for `<initialize>` to determine the state in which the runtime Oracle Configurator opens. See Setting Parameters, page 9-4 for details.

Responsibilities of the Host Application

The responsibilities of the host application for initializing and integrating the runtime Oracle Configurator are:

- Providing end users with a means (such as a Configure button) of posting the initialization message to the Oracle Configurator Servlet. See Setting Parameters, page 9-4 for details.
- Handling initialization of the runtime Oracle Configurator, to prepare it for your user's configuration session. See Invocation of Oracle Configurator by Host Application, page 2-4 for background.
- Disabling visible functions in the surrounding host application that would confuse the user while interacting with the runtime Oracle Configurator.
- Handling the output from the return URL (as described in Return URL Parameter, page 9-14), and closing the configurator window by resetting its frame's location property.
- Handling termination of the runtime Oracle Configurator, to return control and results to the host application when your user closes the window. See Definition of Session Termination, page 10-2 for background.

You may be able to provide your host application with improved performance by preloading the Oracle Configurator Servlet, which involves providing an initialization message in a text file. The name of the text file is specified with the OC Servlet property `cz.uiservlet.pre_load_filename`, as described in the *Oracle Configurator*

Installation Guide. For details on preloading with an initialization message, see the *Oracle Configurator Performance Guide*.

Setting Parameters

You specify <initialize> and its parameters as the value of an XML message that is passed to the Oracle Applications Framework, as described in Invocation of Oracle Configurator by Host Application, page 2-4. The Oracle Applications Framework is called through the URL specified in the profile option BOM: Configurator URL of UI Manager. See the *Oracle Configurator Installation Guide* for details about setting profile options. For more information on the Oracle Applications Framework, see the Oracle Application Framework Documentation Resources, Release 12, in Oracle Applications Documentation, on the Oracle Technology Network.

Parameter Syntax

All parameters to the XML initialization message are specified as name-value pairs, using attributes of the <param> document element, in the form:

Example

```
<param name="parameter_name">parameter_value</param>
```

Syntax of initialization message in HTML context, page 9-4 shows the basic syntax for specifying the Oracle Configurator Servlet's URL and the initialization message as you would typically use them in your host application. The parts that you need to modify are typographically emphasized.

Syntax of initialization message in HTML context
Example

```
...
<script language="javascript" >
function init() {document.test1.submit();}
</script>
<body onload="init();">
<form
action="URL_of_OC_Servlet"
method="post" id="test1" name="test1">
<input type="hidden" name="XMLmsg" value=
'<initialize>
<param name="parameter_1_name">parameter_1_value</param>
<param name="parameter_n_name">parameter_n_value</param>
</initialize>'>
</form>
</body>
...
```

When a Web page containing the kind of HTML coding shown in Syntax of initialization message in HTML context, page 9-4 is rendered in a browser, the initialization message is posted to the URL of the Oracle Configurator Servlet, as described in Invocation of Oracle Configurator by Host Application, page 2-4.

See Basic XML initialization parameters, page 9-6 for some typical values for the parameters, and Minimal HTML for invoking the Runtime Oracle Configurator, page 9-

for a test page that puts the values in context.

- Be aware that XML permits you to use either single or double quotation marks around the value of an element's attribute, so you might also write:

  Example

  ```
  "<initialize>
      <param name='parameter_name'>parameter_value</param>
  </initialize>"
  ```

- XML messages are, by default, case-sensitive. The names of initialization parameters (shown as *parameter_name* in the syntax examples in this chapter) are also case-sensitive. If you pass a custom initialization parameter named `MyParam`, but your code tests for a parameter named `myparam`, then your test will fail.

- You can only insert a given parameter once in the initialization message. If you insert the same parameter more than once, the last occurrence of the parameter is processed, and any preceding occurrences are ignored. This is important to remember when you specify Custom Initialization Parameters in the Configurator Preferences page in Oracle Configurator Developer, as described in the *Oracle Configurator Developer User's Guide*. These custom initialization parameters are prepended to the parameters provided by Configurator Developer itself during a test session. Custom parameters that duplicate Configurator Developer parameters are thus ignored.

- If you need to include non-ASCII characters in your initialization parameters, then specify the required character set as the value of the `charset` parameter in the `meta` element of your HTML page. Several examples follow:

  Example

  ```
  <meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <meta http-equiv="Content-Type" content="text/html; charset=EUC-JP">
  ```

Omitting Parameters or Values

If you omit a parameter entirely from the initialization message, then the parameter is ignored by the runtime Oracle Configurator.

However, if a parameter has a default value, then you must either accept the effect of the default, or override the default with a specified value. The default values for the parameters are provided in Initialization Parameter Descriptions, page 9-17.

> **Note:** If you include a parameter in the initialization message, do not leave its value empty. Doing so causes an error when the initialization message is processed. If you omit the value of a parameter, then the runtime Oracle Configurator generates an error message indicating which parameter is missing a value. The message appears in the

browser window, and in the servlet's session log.

Typical Parameter Values

The example Basic XML initialization parameters, page 9-6 shows an example of a basic set of initialization parameters, illustrating the types of responsibility shown in Types of Initialization Parameters, page 9-9.

See Syntax of initialization message in HTML context, page 9-4 for the syntax of the initialization message, and Minimal HTML for invoking the Runtime Oracle Configurator, page 9-8 for a test page that puts the values in context.

For the complete list of valid initialization message parameters, see Initialization Parameter Descriptions, page 9-17.

Basic XML Initialization Parameters Example

```
<initialize>
  <param name="database_id">serv02_sid01</param>
  <param name="user">operations</param>
  <param name="pwd">welcome</param>
  <param name="calling_application_id">708</param>
  <param name="responsibility_id">22713</param>
  <param name="ui_def_id">9740</param>
  <param name="ui_type">JRAD</param>
  <param name="return_url
">http://www.mysite.com:8802/OA_HTML/myorg/myservlets/Checkout</param>
</initialize>
```

The Explanation of initialization parameters , page 9-6 table explains the parameters used in the example Basic XML initialization parameters, page 9-6.

The following table explains the basic initialization parameters.

Explanation of initialization parameters in Basic XML Initialization Parameters, page 9-6

Parameter type	Name	Description
Login	database_id, page 9-25	The DBC file that identifies the login database.
Login	user, page 9-37	The user ID of the login user.
Login	pwd, page 9-32	The password of the login user.
Login	calling_application_id, page 9-20	The ID of the host application.

Parameter type	Name	Description
Login	responsibility_id, page 9-32	The responsibility of the login user.
Configuration	ui_def_id, page 9-36	The ID of the UI of the model to be configured.
Configuration	ui_type, page 9-36	The type of the UI identified by ui_def_id, page 9-36.
Return	return_url, page 9-33	The URL of the return URL servlet.

Minimal Test of Initialization

Minimal HTML for invoking the Runtime Oracle Configurator, page 9-8 shows the HTML for a minimal web page that calls the runtime Oracle Configurator. combines the invocation of the runtime Oracle Configurator shown in Syntax of initialization message in HTML context, page 9-4 with the initialization message parameters shown in Basic XML initialization parameters, page 9-6. (For simplicity, omits the `return_url` parameter, which is shown in HTML for Invoking the Runtime Oracle Configurator with Return URL, page 9-14.)

You can use this test page as a stand-in for your host application, by opening it in a browser. You must substitute your own site-specific values for the parameters `database_id` and `ui_def_id`. You must also provide a site-specific host name and port for the `action` attribute of the `form` element in Minimal HTML for invoking the Runtime Oracle Configurator, page 9-8.

Minimal HTML for invoking the Runtime Oracle Configurator

```
<html>
<head>
<title>Minimal Configurator Test</title>
</head>
<script language="javascript" >function init()
{document.test1.submit();}</script>
<body onload="init();">
<form
action="http://www.mysite.com:8802/OA_HTML/configurator/UiServlet"
method="post" id="test1" name="test1">
<input type="hidden" name="XMLmsg" value=
'<initialize>
  <param name="database_id">serv02_sid01</param>
  <param name="user">operations</param>
  <param name="pwd">welcome</param>
  <param name="calling_application_id">708</param>
  <param name="responsibility_id">22713</param>
  <param name="ui_def_id">9740</param>
  <param name="ui_type">JRAD</param>
</initialize>'>
</form>
<pre>Loading... </pre></body>
</html>
```

Parameter Validation

When your host application calls the runtime Oracle Configurator, the Oracle Configurator Servlet validates the parameters of the initialization message.

- There must be a way of connecting to the database, such as the parameter database_id, page 9-25.
- There must be a way to choose a Model to be configured, so the initialization message must include one of the combinations described in Model Identification Parameters, page 9-10.
- If there is an error processing the initialization message, the results are posted to the URL specified in the `return_url` parameter.

Initialization parameters are accessible to Configurator Extensions and custom applications that use the Configuration Interface Object (CIO), by calling the method `Configuration.getUserParameters()`, which is described in the *Oracle Configurator Extensions and Interface Object Developer's Guide*.

Logging of Parameter Use

To determine exactly which values of the initialization parameters were used in a configuration session, you can examine the configuration session log files for the Oracle Configurator Servlet. For more information on logging, see Troubleshooting, page 1-11.

Initialization Parameter Types

This section describes the use of the types of initialization parameters listed in the Types of Initialization Parameters, page 9-9 table. All of the initialization parameters are described alphabetically in Initialization Parameter Descriptions, page 9-17.

Types of Initialization Parameters

Type	Required ?	Description	See
Login	Yes	Information required for access to the proper data, such as database, user, and password.	Login Parameters, page 9-9
Configuration	Yes	Identification of the Model to be configured, or of the existing configuration to be modified.	Model Identification Parameters, page 9-10
Publication	Yes, for published models	Information required to select the correct Model publication.	Model Publication Identification Parameters, page 9-13
Return	No, but recommended	Identification of the return URL that handles the results from the runtime Oracle Configurator, such as configuration outputs.	Return URL Parameter, page 9-14
Pricing and ATP	No	Identification of the procedures and interfaces to be used for obtaining prices and ATP dates.	Pricing Parameters, page 9-15 ATP Parameters, page 9-15
Other	No	Miscellaneous information.	Arbitrary Parameters, page 9-16

Login Parameters

To connect the runtime Oracle Configurator to the database, your initialization message must specify one of the combinations of parameters listed in Initialization Parameters Required for Login, page 9-10.

For descriptions of the individual parameters, see Initialization Parameter Descriptions, page 9-17.

Initialization Parameters Required for Login

Parameter Combination	Used to Launch Oracle Configurator From ...
database_id, icx_session_ticket, and responsibility_id	• A host application, using Oracle Applications login authentication • Oracle Configurator Developer, by using the **Test Model** button
database_id, calling_application_id, responsibility_id, user, and pwd	• A stand-alone test page (such as that shown in Minimal HTML for invoking the Runtime Oracle Configurator, page 9-8) • A custom Web application that does not use Oracle Applications login authentication. (In this case, Oracle Configurator constructs an ICX session ticket from the values provided for user, page 9-37, pwd, page 9-32, calling_application_id, page 9-20, and responsibility_id, page 9-32.)

You can use the same set of login parameters for both legacy and generated (HTML-based) UIs. If you do, use the ui_type, page 9-36 parameter to distinguish between the UI types.

Model Identification Parameters

There are several different ways in which you can identify the Model to be configured, or the existing configuration to be modified. In your initialization message, you must use one of the parameters or a combination of the parameters listed in Model Identification Parameters, page 9-10:

Model Identification Parameters

Method for Configuration Identification	Initialization Parameters	Described in ...
User Interface	ui_def_id, page 9-36	Identifying the User Interface Definition, page 9-11
Configuration	config_header_id, page 9-24 config_rev_nbr, page 9-25	Identifying the Configuration, page 9-11

Method for Configuration Identification	Initialization Parameters	Described in ...
Model	For Imported BOM Models: • operating_unit_org_id, page 9-29 or organization_id, page 9-29 • inventory_item_id, page 9-26 For Models created in Configurator Developer: • product_id, page 9-31	Identifying the Model, page 9-12

For detailed descriptions of the individual parameters, see Initialization Parameter Descriptions, page 9-17.

Identifying the User Interface Definition

Parameter to specify:

- ui_def_id, page 9-36

Using this parameter creates a new configuration. It is most useful for identifying a Model created entirely in Oracle Configurator Developer. It is also useful for specifying a particular UI out of several that may be available for a Model, whether or not the Model was created entirely in Configurator Developer.

This ID identifies a User Interface created in Configurator Developer. The User Interface includes identification of the Model to be configured (which is associated with configuration rules).

Identifying the Configuration

Parameters to specify:

- config_header_id, page 9-24
- config_rev_nbr, page 9-25

Using this combination of parameters restores an existing saved configuration, and thus also the model used to create the configuration.

The Configuration Header ID is the main identifier of an existing configuration record previously created and saved by your host application or another application that

knows how to save configurations to the CZ schema, such as the runtime Oracle Configurator. The Configuration Revision Number distinguishes among particular saved configurations sharing the same header information.

Identifying the Model

The parameters you should use to identify the configuration model depend on whether the model is an imported BOM Model or a Model created in Configurator Developer.

Imported BOM Models

Parameters to specify:

- operating_unit_org_id, page 9-29 or organization_id, page 9-29
- inventory_item_id, page 9-26

Using this combination of parameters creates a new configuration. It is only useful for identifying a Model that was originally created in another application (such as Oracle Applications Bills of Materials) and then imported into Oracle Configurator Developer.

Your host application must determine which Model to configure and be able to identify it by Inventory Item ID and Organization ID. See the individual descriptions of these parameters for more detail.

For backward compatibility only, you may need to specify these parameters:

- context_org_id, page 9-25 instead of organization_id, page 9-29
- model_id, page 9-27 instead of inventory_item_id, page 9-26

Models Created in Configurator Developer

Parameters to specify:

- product_id, page 9-31
- config_effective_usage_id, page 9-23 (for custom applications only)
- publication_mode, page 9-32 (for custom applications only)

If the root of your configuration model is a Model that you created in Oracle Configurator Developer, and you entered a Product ID when you published the Model, then you should specify only the product_id, page 9-31 in your initialization message to identify the Model to configure. See the *Oracle Configurator Developer User's Guide* for details about publishing Models.

The use of the Product ID to identify the Model requires the additional specification of the Usage and Mode for publication, according to the following conditions:

- If the host application is a custom application (that is, not part of Oracle Applications), then you must also pass publication_mode, page 9-32 and

config_effective_usage_id, page 9-23 in the initialization message.

- If you do not pass config_effective_usage_id, page 9-23, then Oracle Configurator uses the default value of this parameter, which is `Any Usage`.

- If you do not pass publication_mode, page 9-32, then Oracle Configurator uses the default value of this parameter, which is `P` (Production mode).

- If the host application is part of Oracle Applications (such as Order Management), then Oracle Configurator automatically obtains the Usage and Mode from the profile options CZ: Publication Usage and CZ: Publication Lookup Mode and applies the values to the configuration session. Consequently, you do not have to specify the parameters yourself.

Model Publication Identification Parameters

If your Model has been published, then you need to identify the specific Model publication that you want to configure. This requires that you specify publishing applicability parameters in your initialization message, in addition to those that identify the Model (which are described in Model Identification Parameters, page 9-10).

To determine the Model publication to display, you must specify in your initialization message one or more of the applicability parameters listed in Initialization Parameters for Publishing Applicability, page 9-13. These initialization parameters correspond to the applicability parameters that you specify when creating the publication in the Publications area of the Repository in Oracle Configurator Developer. See Publishing Configuration Models, page 16-1 and the *Oracle Configurator Developer User's Guide* for more information about publishing.

The following table lists the publishing initialization parameters as they appear in the initialization message and in Configurator Developer.

Initialization Parameters for Publishing Applicability

Initialization Parameter	OCD Publishing Parameter
calling_application_id, page 9-20	Applications
config_effective_usage_id, page 9-23	Usages
config_model_lookup_date, page 9-24	Valid From/Valid To
publication_mode, page 9-32	Mode

Support of Multiple Instantiation

This following parameter indicates whether a host application supports multiple instantiation:

- sbm_flag, page 9-34

At runtime, Oracle Configurator checks this flag to see if the host application supports multiple instantiation. If this parameter is present in the initialization message, the model is launched regardless of its type. If the parameter is not present, users are prevented from working with the PTO model and its references to the BOM models under the root model. A message is returned informing the end user that the host application does not support multiple instantiation.

Return URL Parameter

The return URL is the fully qualified URL of a Java servlet installed on your Web server that implements the behavior that you want to invoke after the user has ended a configuration session. The return URL for a configuration session is specified by the following initialization parameter:

- return_url, page 9-33

The example Initialization Messagefor Invoking Oracle Configurator with Return URL, page 9-14 shows the use of this parameter in an initialization message, to specify an example servlet class `myorg.myservlets.Checkout`. That example class is described in Implementing a Return URL Servlet, page E-3

Initialization Message for Invoking Oracle Configurator with Return URL

```
<initialize>
  <param name="database_id">serv02_sid01</param>
  <param name="user">operations</param>
  <param name="pwd">welcome</param>
  <param name="calling_application_id">708</param>
  <param name="responsibility_id">22713</param>
  <param name="ui_def_id">9740</param>
  <param name="ui_type">JRAD</param>
<param
name="return_url">http://www.mysite.com:8802/OA_HTML/myorg/myservlets/Ch
eckout</param>
</initialize>
```

The URL specification in the `return_url` parameter must stop at the name of the servlet class. You cannot pass parameters to the class in this URL (for example, with the *`classname`*?*`parameter`*=*`value`* syntax). The return URL servlet should only get data from the termination message, which is passed to it as the value of the `XMLmsg` argument.

The termination message is sent to the return URL when a configuration session is terminated. This occurs in the event of normal termination, cancellation by the end user, or exceptions.

See The Return URL, page 10-13 for details on the implementation of the return servlet.

Pricing Parameters

These parameters are used when the runtime Oracle Configurator calls existing APIs to get pricing data for configured items.

Because these parameters are designed to be used with an interface using callback procedures, they are also referred to as **callback pricing parameters**.

This guide assumes that you are using Oracle Applications Release 12 and Oracle Advanced Pricing (QP), or your own callback pricing procedures that call Oracle Advanced Pricing.

To use callback pricing, provide the following set of parameters in your initialization message:

- pricing_package_name, page 9-31
- configurator_session_key, page 9-25
- operating_unit_org_id, page 9-29
- either price_mult_items_proc, page 9-30, price_mult_items_mls_proc, page 9-30, or price_single_item_proc, page 9-30

For descriptions of the individual parameters, see Initialization Parameter Descriptions, page 9-17.

See Pricing and ATP in Oracle Configurator, page 13-1 for details on the use of these parameters. See Pricing and ATP Callback Procedures, page E-2 for examples of procedures that might be specified by these parameters.

ATP Parameters

These parameters are used when the runtime Oracle Configurator calls existing APIs to get ATP (Available To Promise) data for configured items.

Because these parameters are designed to be used with an interface using callback procedures, they are also referred to as **callback ATP parameters**.

This guide assumes that you are using Oracle Applications Release 12.

To use callback ATP, provide these parameters:

- atp_package_name, page 9-20
- configurator_session_key, page 9-25
- get_atp_dates_proc, page 9-26

- operating_unit_org_id, page 9-29
- requested_date, page 9-32 (optional, defaults to SYSDATE)
- warehouse_id, page 9-37
- and one of the following:
  - customer_id, page 9-25 *and* customer_site_id, page 9-25
  - ship_to_org_id, page 9-35

For descriptions of the individual parameters, see Initialization Parameter Descriptions, page 9-17.

See Pricing and ATP in Oracle Configurator , page 13-1for details on the use of these parameters. See Pricing and ATP Callback Procedures, page E-2 for examples of procedures that might be specified by these parameters.

Arbitrary Parameters

You can use the `<param>` document element to send arbitrary parameters that are not already provided, or that may be required for particular applications. You specify the arbitrary parameter as a name-value pair, using the syntax described in Parameter Syntax, page 9-4:

Example

```
<param name="parameter_name">parameter_value</param>
```

For example:

Example

```
<param name="org_home_page">http://www.oracle.com</param>
```

Such arbitrary parameters are not processed by the UI Server, but are passed to the Oracle Configuration Interface Object (CIO), thus making them available to Configurator Extensions. See the *Oracle Configurator Extensions and Interface Object Developer's Guide* for information about obtaining a list of the initialization parameters passed.

While the architecture of Oracle Configurator allows for the possibility of validating XML parameters against a DTD, this is not currently enforced.

Parameter Compatibility

Initialization parameters are backwardly compatible. A host application can continue to use the initialization message parameters provided for a previous release with the same results, unless a parameter has been replaced or withdrawn, thus making it obsolete.

Obsolete parameters in the initialization message are ignored by Oracle Configurator. Your host application does not need to remove these parameters from the initialization

message, but they have no effect on the initialization of Oracle Configurator.

Initialization Parameter Descriptions

This section lists alphabetically all the parameters of the initialization message. The use of parameters in the initialization message is described in Setting Parameters, page 9-4. The parameters are summarized in Initialization Parameters for Oracle Configurator, page 9-17.

Initialization Parameters for Oracle Configurator

Name
alt_database_name, page 9-19
application_id, page 9-20
apps_connection_info, page 9-20
atp_package_name, page 9-20
calling_application_id, page 9-20
client_header, page 9-21
client_line, page 9-22
client_line_detail, page 9-22
config_creation_date, page 9-22
config_effective_date, page 9-23
config_effective_usage, page 9-23
config_effective_usage_id, page 9-23
config_header_id, page 9-24
config_model_lookup_date, page 9-24
config_rev_nbr, page 9-25

Name
configurator_session_key, page 9-25
context_org_id, page 9-25
customer_id, page 9-25
customer_site_id, page 9-25
database_id, page 9-25
get_atp_dates_proc, page 9-26
icx_session_ticket, page 9-26
inventory_item_id, page 9-26
jrad_standalone, page 9-26
model_id, page 9-27
model_quantity, page 9-27
operating_unit_org_id, page 9-29
organization_id, page 9-29
price_mult_items_mls_proc, page 9-30
price_mult_items_proc, page 9-30
price_single_item_proc, page 9-30
pricing_package_name, page 9-31
product_id, page 9-31
publication_mode, page 9-32
pwd, page 9-32

Name
read_only, page 9-32
requested_date, page 9-32
return_url, page 9-33
save_config_behavior, page 9-33
share_dio, page 9-34
sbm_flag, page 9-34
ship_to_org_id, page 9-35
template_url, page 9-35
terminate_id, page 9-35
terminate_msg_behavior, page 9-35
ui_def_id, page 9-36
ui_type, page 9-36
user, page 9-37
user_id, page 9-37
warehouse_id, page 9-37

alt_database_name

A fully specified JDBC connect string or URL, specifying the JDBC driver and the database alias of the database to connect to.

This parameter is recommended for use during development of your application, as an alternative to connecting as an Oracle Applications user. It is not recommended for production deployment. To provide security in a production deployment, you can disable this parameter by setting the OC Servlet property `cz.uiserver.allow_alt_database_login` to `false`. This setting prevents a login that uses this parameter. For details on setting this property, see the current release or

patch information for Oracle Configurator on the Oracle Support Web site.

This login parameter is retained for backward compatibility. It is only valid for legacy Oracle Configurator User Interfaces, not for generated User Interfaces (HTML-based). It must be accompanied by user, page 9-37 and pwd, page 9-32.

You must specify thin drivers in the connect string, as shown in the following example.

Example for alt_database_name

```
jdbc:oracle:thin:@server01:1521:vis11
```

application_id

The ID from FND_APPLICATION.APPLICATION_ID that is the ID of the host application.

apps_connection_info

If Oracle Configurator is running in one database (for example, Release 12), and connecting to another database to perform pricing, this parameter describes how to connect to the other database. The `apps_connection_info` element can contain one of the following parameters or sets of parameters:

- database_id, page 9-25
- database_id, page 9-25 and icx_session_ticket, page 9-26
- user, page 9-37, pwd, page 9-32
- alt_database_name, page 9-19, user, page 9-37, and pwd, page 9-32

atp_package_name

The name of the PL/SQL interface package that the runtime Oracle Configurator calls to get ATP information. This parameter is required if the ATP callback interface is to be used. The particular procedure in the package to be used for calculating ATP dates is specified by get_atp_dates_proc, page 9-26.

calling_application_id

The ID obtained from FND_APPLICATION.APPLICATION_ID that identifies the host application. The predefined APPLICATION_ID for Oracle Configurator is 708.

When publishing Models from Oracle Configurator Developer, you must select at least one application from the list of all registered applications. Applications that are not part of Oracle Applications must be registered in Oracle Applications before they can use this parameter. For more information about registering applications, see the *Oracle E-Business Suite System Administrator's Guide*.

If the host application is part of Oracle Applications (for example, Order Management, *i* Store, or TeleSales), it is important to note that the host application displays the publication only if:

- The publication's Application applicability parameter includes the short name of the application (for example, ONT is the short name for Oracle Order Management)
- The application is assigned to the end user's *Responsibility*, which is defined in Oracle Applications

  An Oracle Applications user can often choose one of many Responsibilities, but each Responsibility is assigned to only one application.

You specify applicability parameters when defining a publication in Configurator Developer. For more information, see the *Oracle Configurator Developer User's Guide*.

When the publication is created, a value for FND_APPLICATION.APPLICATION_ID is saved in the database. It is very important to ensure that if the development and production publications are on separate servers, then the custom application must be registered on both servers; it is your responsibility to verify that the custom application's ID is the same on both servers.

See also responsibility_id, page 9-32.

This is a required parameter.

> **Note:** Oracle Order Management (OM) launches Oracle Configurator using a calling application ID in the initialization message that is based on the Responsibility selected by the OM user. When a user accesses the Sales Order form using the "Order Management Super User" responsibility, all configuration models that were published with the "ONT" Publication applicability parameter can be configured using Oracle Configurator. However, when accessing the Sales Order form using the Manufacturing and Distribution Manager responsibility, these same published models are not available, even though the host application is the same (OM). To make the same models available to the Manufacturing and Distribution Manager responsibility, republish your models such that they are also available to Oracle Manufacturing (appears as "Manufacturing" in Configurator Developer).

client_header

A string or number identifying the unit of work for the host application (for example, an order or quote). Used in conjunction with the methodology for input configuration attributes, which is described in the *Oracle Configurator Methodologies* documentation. See also client_line, page 9-22 and client_line_detail, page 9-22.

client_line

A string or number identifying the particular part of the order or quote that the configuration is initiated against. Used in conjunction with the methodology for input configuration attributes, which is described in the *Oracle Configurator Methodologies* documentation. See also client_header, page 9-21 and client_line_detail, page 9-22.

client_line_detail

A string or number used to provide additional information if client_line, page 9-22 does not provide enough. Used in conjunction with the methodology for input configuration attributes, which is described in the *Oracle Configurator Methodologies* documentation. See also client_header, page 9-21 and client_line, page 9-22.

config_creation_date

The host application's notion of when the configuration is created.

The value for the `config_creation_date` parameter must be determined by your host application. It is the host application's notion of when the configuration was created.

See also: config_effective_date, page 9-23 and config_model_lookup_date, page 9-24.

Oracle Order Management specifies a value for this parameter when invoking Oracle Configurator, using by default the value of Model Line Creation Date. The values of config_effective_date, page 9-23 and config_model_lookup_date, page 9-24 are defaulted.

The value of this parameter must be in the format `MM-DD-YYYY-HH-MM-SS`. The values for the tokens in this format are shown in Date and Time Format for Parameter, page 9-22:

The following table lists the date and time formats used for the config_creation_date parameter.

Date and Time Format for config_creation_date, page 9-22 Parameter

Token	Meaning
MM	The number of the month
DD	The number of the day of the month
YYYY	The year

Token	Meaning
HH	The 24-hour representation of the hour
MM	The number of minutes
SS	The number of seconds

Default for config_creation_date
For a new configuration: the value of SYSDATE. For a restored configuration: the saved value of config_creation_date, page 9-22. If the parameter value does not include the `HH-MM-SS` portion, then the default time is assumed to be midnight (`00-00-00`).

Example for config_creation_date
```
<param name="config_creation_date">03-25-2001-19-30-02</param>
```

config_effective_date

The date used to filter effective nodes and rules.

This parameter has the same structure as config_creation_date, page 9-22.

See also config_creation_date, page 9-22 and config_model_lookup_date, page 9-24.

This parameter is not required.

Default for config_effective_date
For a new configuration: the value of config_creation_date, page 9-22. For a restored configuration: the saved value of config_effective_date, page 9-23.

config_effective_usage

This parameter is now deprecated. If you are implementing Multiple Language Support (MLS), you should instead use config_effective_usage_id, page 9-23. You cannot use both parameters together.

The name of a Usage created in Oracle Configurator Developer. Usage names are used to identify publications, and to set the Usage on a runtime configuration session. See the *Oracle Configurator Developer User's Guide* for details.

The value is not case-sensitive.

Default for config_effective_usage
The default value of this parameter is `Any Usage`.

config_effective_usage_id

The numeric ID associated with a Usage created in Oracle Configurator Developer, where you specify a Name and Description for the Usage. Usage IDs are used to

identify publications, and to set the Usage on a runtime configuration session. See the *Oracle Configurator Developer User's Guide* for details.

The Usage ID is stored in the database column MODEL_USAGE_ID in the table CZ_MODEL_USAGES. For more information, see Usages, page 16-9.

To obtain the ID value for this parameter, run the following query, then choose the MODEL_USAGE_ID value that corresponds to the NAME or DESCRIPTION of the desired Usage.

Query for Usage IDs

```
SELECT
 cz_model_usages.MODEL_USAGE_ID,
 cz_model_usages.NAME,
 cz_model_usages_tl.DESCRIPTION,
 cz_model_usages_tl.LANGUAGE,
 cz_model_usages_tl.SOURCE_LANG
FROM
 cz_model_usages,
 cz_model_usages_tl
WHERE
 cz_model_usages_tl.LANGUAGE = USERENV ('LANG')
AND
 cz_model_usages.MODEL_USAGE_ID=cz_model_usages_tl.MODEL_USAGE_ID
AND
 cz_model_usages.in_use = '1';
```

This parameter determines the publishing Usage name for the configuration model. See Models Created in Configurator Developer, page 9-12 for more information about using this parameter.

This parameter replaces config_effective_usage, page 9-23. You cannot use both parameters together.

This parameter is not required.

Default for config_effective_usage_id

The default value of this parameter is `Any Usage`, whose MODEL_USAGE_ID is normally `-1`.

config_header_id

The identifier for an existing configuration. Only used for retrieving a configuration previously saved by the runtime Oracle Configurator. Not present if the configuration was not saved.

The value for the `config_header_id` parameter is obtained from CZ_CONFIG_HDRS.CONFIG_HDR_ID in the CZ schema.

config_model_lookup_date

Date to look up the publication for the configuration Model. This parameter has the same structure as config_creation_date, page 9-22.

See also: config_effective_date, page 9-23 and config_model_lookup_date, page 9-24.

This parameter is not required.

Default for config_model_lookup_date

For a new configuration: the value of config_creation_date, page 9-22. For a restored configuration: the saved value of config_effective_date, page 9-23, or SYSDATE, as determined by RestoredConfigDefaultModelLookupDate in CZ_DB_SETTINGS; see RestoredConfigDefaultModelLookupDate, page 4-23 for details.

config_rev_nbr

The configuration revision number. Only used for retrieving a configuration previously saved by the runtime Oracle Configurator. Not present if the configuration was not saved.

The value for the `config_rev_nbr` parameter is obtained from CZ_CONFIG_HDRS. CONFIG_REV_NBR in the CZ schema.

configurator_session_key

An application-dependent string that identifies a configuration session, and allows linking a pricing or ATP request from the runtime Oracle Configurator to the host application entity that started the configuration session. Examples for creating this key might be: order header ID with order line ID, or quote ID with quote revision number.

context_org_id

This parameter is for backward compatibility only. Instead of this parameter you should use its synonym, organization_id, page 9-29.

This parameter is the organization identifier for the BOM exploder. The value for the `context_org_id` parameter must be determined by your host application. It is ultimately derived from MTL_SYSTEM_ITEMS.ORGANIZATION_ID.

customer_id

When getting ATP dates, the ID of the customer to which the configured product is to be shipped. Must be used with customer_site_id, page 9-25.

customer_site_id

When getting ATP dates, the ID of the customer site to which the configured product is to be shipped. Must be used with customer_id, page 9-25.

database_id

The name of the DBC file that contains database connectivity information, without its filename extension of `.dbc`. This file can be found in a standard Oracle Applications installation by calling the PL/SQL function `fnd_web_config.database_id`. This

parameter must be used with certain other parameters, as described in Login Parameters, page 9-9.

Example for database_id
`myhost01_mysid05`

get_atp_dates_proc

The name of the "get ATP dates" procedure to be called from the package specified by atp_package_name, page 9-20. This parameter is conditionally required; it must be provided if the ATP callback interface is to be used.

icx_session_ticket

An ICX session ticket encodes an Oracle Applications session.

This is the recommended way for Oracle Applications to call the runtime Oracle Configurator.

You can use the PL/SQL function `cz_cf_api.icx_session_ticket` to obtain a value for this parameter. (See the description of ICX_SESSION_TICKET, page 17-53 for details about the function `cz_cf_api.icx_session_ticket`.)

When passing an `icx_session_ticket`, the host application must also pass a database_id, page 9-25.

inventory_item_id

This parameter is a synonym that replaces model_id, page 9-27.

This parameter is the imported Inventory Item ID for the top-level imported BOM Model. It is used together with organization_id, page 9-29 to identify the configuration model. The value for this parameter must be determined by your host application. It is ultimately derived from MTL_SYSTEM_ITEMS.INVENTORY_ITEM_ID.

This parameter is conditionally required. There is no default value.

jrad_standalone

Controls whether the user interface for the runtime Oracle Configurator is designed to stand alone in its own window, or to be part of its host application's window. The standalone design includes the page header and global buttons provided by the Oracle Applications Framework. For more information about the Oracle Applications Framework, see the Oracle Application Framework Documentation Resources, Release 12, on Oracle Applications Documentation, on the Oracle Technology Network.

The values allowed for this parameter are shown in the following table:

Value	Meaning
`true`	The UI for the runtime Oracle Configurator is rendered with a header and global buttons.
`false`	The UI for the runtime Oracle Configurator is rendered without a header or global buttons.

Default for jrad_standalone

The default value of this parameter is `false`.

model_id

This parameter is for backward compatibility only. Instead of this parameter you should use its synonym, inventory_item_id, page 9-26.

This parameter is the inventory item identifier for the top-level Model.

The value for the `model_id` parameter must be determined by your host application. It is ultimately derived from MTL_SYSTEM_ITEMS.INVENTORY_ITEM_ID.

Conditionally required. No default.

model_quantity

Only BOM Models can be configured with this parameter. The value of this parameter is a number that indicates how many identical copies of the Model are being configured. The model quantity may change during a configuration session, so the final quantity should be read from the associated output item in the termination message.

Default for model_quantity

For a new configuration, the default is 1. The host application may set a different number.

Notes

Be aware of the effect of passing various values for this parameter when:

- The model is a BOM Model. (Only BOM Models can be configured with the model_quantity, page 9-27 parameter.)
- There exist configuration rules that contribute some quantity to the numeric value of the model root (that is, the rules specify that a certain quantity of the model should be in the configuration).

  Background: Only rules defined on non-BOM nodes can make such contributions. Otherwise, Quantity Cascade calculations result in a numeric cycle.

- These rules are triggered when the configuration is created, rather than as the result of user selections.

  Background: A rule is triggered when the conditions defined for it are satisfied.

Examples:

- A BOM Model is modified by adding a Feature with one Option and a Min/Max of (1,1). A Numeric Rule is defined on that Feature which contributes a value to the quantity of the root BOM Model. When a configuration is created, the condition for the rule is satisfied (because a Min/Max of (1,1) results in a mandatory selection of the Option), and the quantity specified by the Numeric Rule is contributed.

- A BOM Model is modified by adding an Integer Feature with an initial value. A Numeric Rule is defined on that Feature, which contributes the value of the Feature to the quantity of the root BOM Model. When a configuration is created, the condition for the rule is satisfied, and the quantity specified by the Numeric Rule is contributed.

The effects of combining contributions to the model's quantity with passing a value for the initialization parameter `model_quantity` when creating or restoring a configuration is illustrated in Effects of Contributions to Model Quantity, page 9-28. Not all of the possible scenarios are illustrated.

In Effects of Contributions to Model Quantity, page 9-28, the following symbols are used:

- C represents a contribution from a configuration rule to the root BOM model that exists at the creation of the configuration.

- NM represents a value for the `model_quantity` parameter that is passed in while creating a new configuration.

- RM represents a value for the `model_quantity` parameter that is passed in while restoring a saved configuration.

The following table lists the effects of contributions to Model Quantity by different values of the model_quantity parameter.

Effects of Contributions to Model Quantity

	Contribution	Model Quantity	Final Quantity
New Configuration:			
Case 1	C	NM>=C	NM

	Contribution	Model Quantity	Final Quantity
Case 2	C	NM<C	C, with Validation FailureThese Validation Failure messages are deleted once their text is viewed.
Case 3	None or 1	None	1
Case 4	C>1	None	C
Restored Configuration:			
Saved In Case 1	C	RM>=C	RM
Saved In Case 1	C	RM<C	C, with Validation Failure
Saved In Case 1	None	RM	RM
Saved In Case 1	C	None	NM

operating_unit_org_id

The Organization ID for the Oracle Applications Operating Unit Organization.

This parameter supports the use of Multiple Organization Access Control (MOAC), which requires the Operating Unit to be provided for a configuration session.

Oracle Applications that call Advanced Pricing or ATP pass this parameter when invoking the Runtime Oracle Configurator. You must provide this parameter in a customized initialization message, or in the Custom Initialization Parameters field in Oracle Configurator Developer, if you use Advanced Pricing or ATP, or use MOAC-enabled code in a Configurator Extension.

When this parameter is provided, Oracle Configurator establishes the provided Operating Unit Organization for the configuration session. If the parameter is not provided, the default Operating Unit Organization is established. If there is no default, then no Operating Unit Organization is established.

organization_id

The Organization ID for the Oracle Applications Item Validation Organization.

This parameter is a synonym that replaces context_org_id, page 9-25.

This parameter is the imported Organization ID for the top-level imported BOM Model. It is used together with inventory_item_id, page 9-26 to identify the configuration model. The value for this parameter must be determined by your host application. It is ultimately derived from MTL_SYSTEM_ITEMS.ORGANIZATION_ID.

If you are using Oracle Order Management, this is the organization identifier for the BOM exploder. See the documentation for Order Management for information on setting the Item Validation organization.

price_mult_items_mls_proc

This is the name of the "price multiple items" procedure to be called in an MLS environment. This parameter should be used by a host application that supports multiple currencies, not just USD (US dollars).

When using pricing callbacks, you must include a parameter that specifies a pricing procedure, such as price_mult_items_mls_proc. However, this parameter is mandatory in an MLS environment. See Pricing Parameters, page 9-15 for background information about parameters that are required for pricing callbacks.

Use this parameter instead of price_mult_items_proc, page 9-30, because the procedure called through this parameter displays prices in the correct currency and format.

price_mult_items_proc

The name of the "price multiple items" procedure to be called from the package specified by pricing_package_name, page 9-31.

This parameter is conditionally required; either this parameter, price_single_item_proc, page 9-30 or price_mult_items_mls_proc, page 9-30 must be provided if pricing callbacks are used.

Use price_mult_items_mls_proc, page 9-30 instead of this parameter, because the procedure called through this parameter displays prices only in USD (US dollars).

This parameter takes precedence over price_single_item_proc, page 9-30.

price_single_item_proc

This parameter is now deprecated. Use price_mult_items_proc, page 9-30 if possible.

The name of the "price single item" procedure to be called from the package specified by pricing_package_name, page 9-31.

This parameter is conditionally required; one of this parameter, price_mult_items_proc, page 9-30, or price_mult_items_mls_proc, page 9-30 must be provided if pricing callbacks are to be used.

This procedure is not called if price_mult_items_proc, page 9-30 is provided.

pricing_package_name

The name of the PL/SQL interface package that the runtime Oracle Configurator calls to get pricing information. This parameter is required if the pricing callback interface is to be used. The particular procedure in the package to be used for performing pricing is specified by either price_mult_items_proc, page 9-30 or price_single_item_proc, page 9-30.

product_id

For a Model created in Configurator Developer, the value for this parameter is the string you enter for Product ID when you create the publication record for the Model.

For an imported BOM Model, the value for this parameter is automatically generated when you create the publication record for the BOM Model (by concatenating the imported Organization ID with the imported Inventory Item ID) and cannot be modified. If you are configuring a BOM Model, you should probably use the combination of organization_id, page 9-29 and inventory_item_id, page 9-26 instead of this parameter.

If this parameter is included in the initialization message, Oracle Configurator uses the function CZ_CF_API.CONFIG_MODEL_FOR_PRODUCT, page 17-19 to determine which Model and User Interface should be used.

The value for this parameter is obtained from CZ_MODEL_PUBLICATIONS. PRODUCT_KEY in the CZ schema.

The use of the Product ID to identify the model requires the additional specification of the Usage and Mode for publication. If the host application is a custom application (that is, not part of Oracle Applications), then you must also pass publication_mode, page 9-32 and config_effective_usage_id, page 9-23. If the host application is part of Oracle Applications (such as Order Management), then the Usage and Mode are obtained directly from the profile options CZ: Publication Usage and CZ: Publication Lookup Mode.

See Models Created in Configurator Developer, page 9-12 for more information about using this parameter.

Default for product_id

This parameter is conditionally required. There is no default value.

Examples for product_id

To make your application use a Configurator Developer Model with the Product ID of ABC1234, insert the following parameter in your initialization message:

```
<param name="product_id">ABC1234</param>
```

To make your application use an imported BOM Model with the organization_id, page 9-29 204 and the inventory_item_id, page 9-26 137, insert the following parameter in your initialization message:

```
<param name="product_id">204:137</param>
```

publication_mode

Determines the publication mode for the configuration model. See Models Created in Configurator Developer, page 9-12 for more information about using this parameter.

The values allowed for this parameter are shown in the following table:

Value	Meaning
P	Production
T	Test

This parameter is not required.

Default for publication_mode
The default value of this parameter is P.

pwd

The password to use when logging in to the Oracle Applications database. Use the Oracle Applications password if you identified the database with the database_id, page 9-25 parameter. Use the database password if you identified the database with the alt_database_name, page 9-19 parameter. Used in conjunction with user, page 9-37.

read_only

If the value is true, the UI Server provides a read-only UI for viewing configurations. The end user can examine options, but cannot select any. The **Finish** button is disabled. The UI Server displays a message at the beginning of the configuration session, indicating that the session is read-only. If the value is false, the UI Server provides the normal UI for configuring a model.

Default for read_only
The default value of this parameter is false.

requested_date

When getting ATP dates, the requested date entered on the order line. The format of the date must be MM-dd-yyyy. The default value of SYSDATE is used if you do not specify a different date.

responsibility_id

When logging in to Oracle Applications, the responsibility determines the functions available to the login user. The value to use for this ID is obtained from

FND_RESPONSIBILITY_VL.RESPONSIBILITY_ID.

The predefined RESPONSIBILITY_ID for the Oracle Configurator Developer responsibility is 22713. The responsibilities related to Oracle Configurator are described in The Predefined Configurator Developer Responsibilities, page 15-2.

See also calling_application_id, page 9-20.

return_url

The fully qualified URL of a Java servlet installed on your Web server that implements the necessary behavior after a configuration session is terminated. See Return URL Parameter, page 9-14 for details, and Implementing a Return URL Servlet, page E-3 for a code example.

The example below shows the use of this parameter to specify a servlet class `myorg.myservlets.Checkout`, which is the example described in Implementing a Return URL Servlet, page E-3

Example for return_url

```
<param
name="return_url">http://www.mysite.com:8802/OA_HTML/myorg/myservlets/Ch
eckout</param>
```

> **Note:** It is necessary to register an alias for the return URL servlet. For details, see the *Oracle Configurator Installation Guide*.

save_config_behavior

The values allowed for this parameter are shown in the following table:

Value	Meaning
`never`	A new configuration is not saved.
`new_config`	A new configuration is saved.
`new_revision`	A new revision of the configuration is saved. (If no existing revision is found, a new configuration is saved.)
`overwrite`	The existing configuration header and revision is used.

Default for save_config_behavior

The default value of this parameter is `new_revision`.

If the value is `overwrite`, an error is signalled.

sbm_flag

This parameter indicates whether the host application supports multiple instantiation. To support multiple instantiation the host application must have the appropriate patch applied. The following table describes the valid values for the `sbm_flag` parameter.

Value	Meaning
`True`	The host application has installed the appropriate software patch that supports multiple instantiation.
`False`	The software patch supporting multiple instantiation has not been installed, and multiple instantiation is not supported by the host application.

A message is returned when an end user attempts to instantiate a component at runtime, and the host application does not support instantiation. If the sbm_flag is not passed at all, host application support of multiple instantiation is considered False.

share_dio

See the description of the related servlet property `cz.uiservlet.dio_share` in the *Oracle Configurator Installation Guide*. This initialization parameter overrides that servlet property, if both are present.

The values allowed for this parameter are shown in the following table.

Value	Meaning
`false`	Disables sharing the cached version of the Model. This provides slower loading of the Model, but reflects the latest changes to the Model.
`true`	Enables sharing the cached version of the Model. This provides faster loading after the initial loading of the Model, but does not reflect the latest changes to the Model.

ship_to_org_id

When getting ATP dates, the ID of the organization to which the configured product is to be shipped. This value is obtained from SHIP_TO_ORG_ID in the OE_ORDER_LINES_ALL table.

template_url

Used only with DHTML legacy user interfaces, which are no longer supported..

terminate_id

Important: As of this release, DHTML UIs are no longer supported.

Identification number used to support guided selling in Oracle Order Management. An **Applet** session running in the UI Server generates a termination ID (which is a sequence number) and inserts it into the initialization message for the DHTML session (also running in the UI Server), as the value of this initialization parameter. When the DHTML session terminates, it stores its XML termination message in the database, identified by this termination ID. The Applet session then uses the termination ID to fetch the XML termination message from the database and return it to the host application (Order Management). For a related subject, see the discussion of the **heartbeat** mechanism and guided selling in the *Oracle Configurator Installation Guide*.

terminate_msg_behavior

The values allowed for this parameter are shown in the following table:

Value	Meaning
`full`	The entire termination message is passed back to the host application. This includes prices, if you have used a pricing interface package (see Pricing and ATP in Oracle Configurator , page 13-1).
`brief`	No output or messages are passed to the caller.

It is recommended that host applications using the CZ_CONFIG_DETAILS_V view to read configuration outputs use `brief` when the configuration is saved. If the configuration is not saved, then the outputs and messages are not readable from the

database. If Oracle Configurator receives a connection error or other error, the error messages that it receives are passed back as messages even if the `terminate_msg_behavior` is brief.

ui_def_id

The identifier for a particular User Interface created in Configurator Developer. The value for the `ui_def_id` parameter is obtained by:

- Examining the **UI ID** column in the User Interface area of the Workbench in Oracle Configurator Developer
- Querying CZ_UI_DEFS.UI_DEF_ID in the CZ schema
- Calling the PL/SQL function `cz_cf_api.ui_for_item` (see UI_FOR_ITEM, page 17-66)

ui_type

Indicates the type of user interface being specified for the model being configured. The type determines the agent that renders the UI in the runtime Oracle Configurator. See Runtime UI Types, page 2-6 for background on the UI types provided by Oracle Configurator.

The values allowed for this parameter are shown in the following table:

Value	Meaning
`Applet`	The UI is a legacy Applet UI.
`Custom`	The UI is a custom JSP UI.
`DHTML`	The UI is a legacy DHTML UI. **Important:** As of this release, DHTML UIs are no longer supported.
`JRAD`	The UI is a generated HTML UI.

The initialization message for all UI types is posted to the Oracle Configurator Servlet. For the `JRAD` type, the UI is rendered by the Oracle Applications Framework. For the Applet type, the UI is rendered by the Oracle Configurator Servlet.

You cannot change the actual type of a UI by changing the value of this parameter.

user

The username to use when logging in. Use the Oracle Applications username if you identified the database with the database_id, page 9-25 parameter. Use the database username if you identified the database with the alt_database_name, page 9-19 parameter. Used in conjunction with pwd, page 9-32.

user_id

The ID from FND_USER.USER_ID.

warehouse_id

When getting ATP dates, the ID of the organization that is going to ship the configured product to the customer. This value is obtained from SHIP_FROM_ORG_ID in the OE_ORDER_LINES_ALL table.

10

Session Termination

This chapter describes the format and parameters of the termination message for the runtime Oracle Configurator Servlet.

This chapter covers the following topics:

- Introduction
- Overview
- XML Message Structure
- Submission
- Cancellation
- Error
- The Return URL

Introduction

This chapter describes the format and parameters of the termination message for the runtime Oracle Configurator, including information on:

Note: If your host application is part of Oracle Applications, then the termination message is already defined. You only need to implement a termination message for custom host applications.

Overview

This section provides an overview of the termination message.

Relationship to Initialization Message

This document describes the role of the termination message primarily in relation to the initialization message, in Session Initialization, page 9-1. See the following sections for details:

- Return URL Parameter, page 9-14
- Responsibilities of the Host Application, page 9-3
- return_url, page 9-33
- terminate_id, page 9-35
- terminate_msg_behavior, page 9-35
- model_quantity, page 9-27

Definition of Session Termination

Sessiontermination takes place when the Oracle Configurator window is closed by one of the conditions listed in Termination conditions, page 10-2 table.

Termination conditions

Condition	Example	Explanation
Submission	Your user clicks the **Finish** button.	See Submission, page 10-4
Cancellation	Your user clicks the **Cancel** button.	See Cancellation, page 10-12
Error	A connection cannot be made to the database.	See Error, page 10-12

When the Oracle Configurator window is closed, terminating your user's configuration session, the OC Servlet returns the results to your host application in the form of a termination message, written in XML. You need to understand the structure of the termination message to be able to extract the necessary data from it in your return URL servlet. The structure of this message is described in XML Message Structure, page 10-3.

XML Message Structure

All outputs in the XML termination message are written as XML elements and subelements of the <terminate> document element, in the general form:

Example

```
<terminate>
  <element_name>element_value</element_name>
  <element_name>
    <subelement_name>subelement_value</subelement_name>
  </element_name>
</terminate>
```

The top-level structure of the <terminate> element is illustrated by these excerpts from its DTD:

Example

```
...
<!ELEMENT terminate (config_header_id?, config_rev_nbr?,
valid_configuration?, complete_configuration?, exit, config_outputs?,
config_messages?)>
...
<!ELEMENT config_outputs (output_option*)>
...
<!ELEMENT config_messages (message*)>
...
```

Structure of Termination Message, page 10-4 shows the basic structure of a sample XML termination message. Typographical emphasis and comments have been added to point out the structure; such comments do not appear in actual termination messages.

Structure of Termination Message

```
<terminate>
  <!-- configuration status elements -->
  <config_header_id>1780</config_header_id>
  <config_rev_nbr>2</config_rev_nbr>
  <valid_configuration>true</valid_configuration>
  <complete_configuration>true</complete_configuration>
  <exit>save</exit>
  <config_outputs>
    <option>
      <component_code>143-1490</component_code>
      <quantity>1</quantity>
      <list_price>0.00</list_price>
      <!-- more elements go here -->
    </option>
    <!-- more options go here -->
  </config_outputs>
  <config_messages>
    <message>
      <message_type>error</message_type>
      <message_text>Config header does not exist in
database.</message_text>
    </message>
    <!-- more messages go here -->
  </config_messages>
</terminate>
```

Submission

Submission occurs after your user closes the Oracle Configurator window by clicking the **Finish** button.

The meaning of the **Finish** button is defined by the context of your host application. For instance, in a web store, it might mean adding the configured product to your user's " shopping cart", or submitting the configured order to your order entry system.

When the **Finish** button is clicked, the OC Servlet determines whether a return URL has been specified. If so, the servlet identified by that URL is called, and the results it generates are passed to your host application for further processing. This is the most important job of the return URL servlet; it captures the configuration selections of your user so that your host application can make use of them. For more details, see The Return URL., page 10-13

After the Oracle Configurator window is closed, your host application must repaint the frame used by the Oracle Configurator window.

After submission, the termination message provides the host application with data describing:

- Configuration Status, page 10-5
- Configuration Outputs, page 10-8
- Configuration Messages, page 10-11

Note: If you are providing guided selling in Oracle Applications Order Management, then your host application should obtain the termination message by using the initialization parameter terminate_id, page 9-35. See the description of that parameter for details.

If a custom host application wraps the runtime Oracle Configurator in its own JavaServer Page (as described in Incorporation of Oracle Configurator in the Host Application's UI, page 2-5), then Oracle Configurator posts the termination message to it by HTTP connections, using the return URL (see The Return URL, page 10-13). An example of such a host application is Oracle *i*Store (IBE).

If an Oracle Applications Framework host application incorporates the runtime Oracle Configurator in a region of its own OA Framework page (as described in Incorporation of Oracle Configurator in the Host Application's UI, page 2-5), then Oracle Configurator leaves the termination message in the `OAPageContext`, identified by the transient session key `czTerminateMessage`, then redirects to the same page. An example of such a host application is Oracle Contracts Core (OKC). Note that the termination message may contain error information (see Error, page 10-12) as well as normal termination output.

The host application can retrieve the termination message from the `OAPageContext`, using the following method, where `pageContext` is an instance of `oracle.apps.fnd.framework.webui.OAPageContext`:

Example

```
(String) pageContext.getTransientSessionValue("czTerminateMessage");
```

Configuration Status

The current configuration status is described by the subelements of `<terminate>` listed in this section. These subelements are:

- config_header_id, page 10-6
- config_rev_nbr, page 10-6
- complete_configuration, page 10-6
- exit, page 10-6
- prices_calculated_flag, page 10-6
- standard_validation, page 10-7
- valid_configuration, page 10-8

Subelements for Configuration Status

This section describes the configuration status subelements of the `<terminate>` element.

config_header_id

The main identifier of an existing configuration. See the description for config_header_id, page 9-24. This value is displayed in the Oracle Configurator window with the default label "Configuration Header ID".

config_rev_nbr

The revision number of an existing configuration. See the description for config_rev_nbr, page 9-25. This value is displayed in the Oracle Configurator window with the default label "Configuration Revision".

complete_configuration

The value is `true` if all mandatory option classes (required features) are satisfied. This value is displayed in the Oracle Configurator window with the default label "Configuration Complete".

exit

The possible values written for the exit termination element are shown in the following table:

Value	Meaning
`save`	If the configuration was saved.
`cancel`	If the configuration was cancelled.
`error`	If an error was detected while executing in the UI Server.
`processed`	If a batch validation message was processed but not saved.

This value is displayed in the Oracle Configurator window with the default label "Exit Status".

prices_calculated_flag

Prices are calculated when the user clicks the **Summary** button. This element tells the host application whether this calculation has happened in synchronization with the configuration. The possible values written for the prices_calculated_flag termination

element and their meanings are shown in the following table:

Value	Meaning
`true`	The configuration has not been changed since the end user clicked the **Summary** button. That is, the calculated prices are still in synchronization with the configuration.
`false`	Prices were not calculated after the configuration had been changed. This could happen if the end user had never clicked the **Summary** button before clicking **Finish**, or if the user changed the configuration and did not click the **Summary** button before clicking **Finish**. In this case, the host application should reprice each configuration line, to ensure that the proper prices are applied to the configuration.

standard_validation

This element is added to the termination message only if:

- The configuration session was for batch validation
- The validation phase of batch validation was skipped

See Skipping Batch Validation, page 11-9 for background.

The following table lists the values allowed for the standard_validation element.

Value	Meaning
`true`	The standard validation phase of batch validation was executed.
`false`	The standard validation phase of batch validation was skipped.

total_price

Contains the total discounted selling price for all the selected items in the configuration. The selling price and discounts are determined by the callback pricing procedure that

you have specified for the configuration session. See Pricing and ATP in Oracle Configurator , page 13-1 for details.

valid_configuration

The value is `true` if no error messages are reported for the configuration. This value is displayed in the Oracle Configurator window with the default label "Configuration Valid".

Configuration Outputs

The list of options selected by your user during the configuration session is contained in the `<config_outputs>` subelement of `<terminate>`. Each option is enclosed in `<option>` tags and contains the elements described in this section. These subelements are:

- atp_date, page 10-9
- atp-rollup-date, page 10-9
- bom_item_type, page 10-9
- bom-quantity, page 10-10
- component_code, page 10-10
- discounted_price, page 10-10
- inventory_item_id, page 10-10
- list_price, page 10-10
- organization_id, page 10-10
- parent_line_id, page 10-10
- quantity, page 10-10
- selection_line_id , page 10-10
- uom, page 10-11

Configuration Outputs in the Termination Message, page 10-9 shows an example of configuration outputs in the termination message, with comments added.

Configuration Outputs in the Termination Message

```
<terminate>
  <!-- configuration status goes here -->
  <config_outputs>
    <option>
      <selection_line_id>1846</selection_line_id>
      <parent_line_id>1847</parent_line_id>
      <component_code>143-1490</component_code>
      <quantity>1</quantity>
      <list_price>0.00</list_price>
      <inventory_item_id>1490</inventory_item_id>
      <organization_id>204</organization_id>
      <uom>Ea</uom>
      <discounted_price>0.00</discounted_price>
      <atp_date></atp_date>
    </option>
    <!-- more options go here -->
  </config_outputs>
  <!-- configuration messages go here -->
</terminate>
```

Subelements for Configuration Outputs

This section describes the subelements for the <config_outputs> subelement of the <terminate> element.

atp_date

Contains the **ATP** date. This is calculated by using the ATP procedure specified in the initialization message. See ATP Parameters, page 9-15, and Pricing and ATP in Oracle Configurator , page 13-1.

atp-rollup-date

Provided if ATP is enabled. Contains the ATP date for the entire model.

bom_item_type

Indicates the type of the configured BOM node, using the values shown in Values for the Termination Message Element <bom_item_type>, page 10-9.

The following table lists the values for the termination message element <bom_item_type>.

Values for the Termination Message Element <bom_item_type>

Value	Name	Meaning
1	BOM_MODEL	BOM Model
2	BOM_OPTION_CLASS	BOM Option Class

Value	Name	Meaning
4	BOM_STD_ITEM	BOM Standard Item

bom-quantity

Contains the quantity of the BOM Model being configured, as of the time that the configuration is saved.

component_code

Contains a value extracted from BOM_EXPLOSIONS.COMPONENT_CODE.

discounted_price

Contains the discounted price for the selected option. This is calculated by using the pricing procedure specified in the initialization message. See Pricing Parameters, page 9-15, and Pricing and ATP in Oracle Configurator , page 13-1.

inventory_item_id

Contains the ID for the item, extracted from MTL_SYSTEM_ITEMS. INVENTORY_ITEM_ID.

list_price

Contains the list price for the selected option. This is calculated by using the pricing procedure specified in the initialization message. See Pricing Parameters, page 9-15, and Pricing and ATP in Oracle Configurator, page 13-1.

organization_id

Contains the organization ID for the item, extracted from MTL_SYSTEM_ITEMS. ORGANIZATION_ID.

parent_line_id

Contains the value from CZ_CONFIG_ITEMS.CONFIG_ITEM_ID for the parent node of the configured node. If the parent is the root node, then the value is 0 (zero).

quantity

Contains the selected quantity for the option.

selection_line_id

Contains the ID of the configuration line. It is the same as CZ_CONFIG_ITEMS. CONFIG_ITEM_ID in the CZ schema.

uom

Contains the unit of measure.

Configuration Messages

The messages generated by the OC Servlet in response to selections made by your user during the configuration session are contained in the <config_messages> subelement of <terminate>. Each message is enclosed in <message> tags and contains the elements described in this section. These subelements are:

- component_code, ps_node_id, page 10-11
- item_name, page 10-11
- message_text, page 10-12
- message_type, page 10-12

See Error, page 10-12 for details on how to handle validation failures.

Configuration Messages in the Termination Message, page 10-11 shows an example of a configuration message in the termination message, with typographical emphasis and comments added.

Configuration Messages in the Termination Message Example

```
<terminate>
  <!-- configuration status goes here -->
  <!-- configuration outputs go here -->

  <config_messages>    <message>      <message_type>error</message_type>
<message_text>Config header does not exist in database.</message_text>
</message>
    <!-- more messages go here -->
  </config_messages>
</terminate>
```

Subelements for Configuration Messages

This section describes the subelements for the <config_messages> subelement of the <terminate> element.

component_code, ps_node_id

If present, one of these elements contains the identifier of the option to which this message is related. May be absent, if the message was not generated by a node.

item_name

Contains the name of the option to which this message is related.

message_text

Contains the text of the message.

message_type

Contains the severity level of the message. Possible values include the following:

- suggestion
- warning
- overridable error
- error
- autoselection
- autoexclusion
- not satisfied

Cancellation

Cancellation occurs after your user closes the Oracle Configurator window by clicking the **Cancel** button. Control is returned to the host application, and no configuration information is returned. Validation failure information is not returned in the termination message for a cancellation. The termination message contains only the `<exit>` subelement, with a value of `cancel`:

Cancellation in the Termination Message

```
<terminate>
  <exit>cancel</exit>
</terminate>
```

Error

Error occurs after some condition prevents initialization of the Oracle Configurator window, or submission of the user's selections. Such conditions might include:

- Incorrect database connection or user login parameters (see Login Parameters, page 9-9)
- Lack of any configuration parameters (see Model Identification Parameters, page 9-10)
- Incorrect type for a parameter
- A fatal exception in the Configurator Messaging service

If there were validation failures during your user's configuration session, each failure on the list of the validation failure objects is returned as a <message> element describing the failure. Information about the failure is returned to the OC Servlet as an object of type oracle.apps.cz.cio.ValidationFailure, which you can access through the Oracle Configuration Interface Object (CIO).See the *Oracle Configurator Extensions and Interface Object Developer's Guide* for details.

Control is returned to the host application, and no configuration information is returned. As shown in Error Information in the Termination Message, page 10-13, any validation failures are returned as messages in the <config_messages> element and the termination message contains the <exit> subelement, with a value of error.

Error Information in the Termination Message

```
<terminate>
  <valid_configuration>false</valid_configuration>
  <complete_configuration>false</complete_configuration>
  <exit>error</exit>
  <config_messages>
    <message>
      <message_type>error</message_type>
      <message_text>Problem processing normal request: Could not post
XML message to result URL:Connection refused</message_text>
  </message>
  </config_messages>
</terminate>
```

The Return URL

The program specified by the return_url initialization parameter determines how your host application uses the configuration information produced by your user's selections in the Oracle Configurator window. For demonstration purposes, the return URL program shown in this document is a Java servlet, but you can use another type of program that performs the same role.

If you have specified the return URL using a parameter in your initialization message for the Oracle Configurator window, then the return URL servlet is called upon termination of a configuration session. For details about this parameter, see Return URL Parameter, page 9-14.

The termination message is passed to the return URL as the value of the XMLmsg argument. The initialization message that was passed to the configurator is also passed to the return URL, as the value of the INITmsg parameter.

The return URL must perform all middle-tier and database processing of the configuration and then return HTML that closes the Oracle Configurator window and continues with the program flow for the host application.

Specifying the Return URL

You specify the location of your return URL servlet in the XML initialization message, as the value of the parameter return_url. For an example, see HTML for Invoking the

Runtime Oracle Configurator with Return URL, page 9-14.

See also:

- The Return URL , page 10-13
- return_url, page 9-33
- Parameter Syntax, page 9-4

Implementing the Return URL

The first step in implementing a return URL is to register an alias name for the return URL servlet. For details, see the *Oracle Configurator Installation Guide.*

An example of a return URL servlet is shown in Example Return URL Servlet (Checkout.java), page E-5. You can modify this servlet code for your host application's requirements.

To use some of the configuration information returned in the termination message (for example, the outputs described in Configuration Outputs, page 10-8), you can write a Java method that obtains the value of an element in the termination message by using the `getTagValue()` method defined in the Checkout servlet.

The following code fragment obtains the value of the `<valid_configuration>` output:

Obtaining Values from Termination Message

```
String getValidConfig(XMLDocument doc) {
   // get element from termination msg
    return getTagValue(doc, "valid_configuration", null);
  }
```

For example, the following value of the `<valid_configuration>` output is provided by the following termination message:

Example

```
<valid_configuration>true</valid_configuration>
```

When the Checkout servlet is called after submission, it replaces the Oracle Configurator window with an HTML page, like this:

HTML Output Produced from Termination Message

```
<html>
<head><title>Checked Out with Valid Configuration</title></head>
<body>
Configuration Valid?: true
</body>
</html>
```

11

Batch Validation

This chapter describes using Oracle Configurator in a programmatic mode.

This chapter covers the following topics:

- Overview
- Introduction
- Passing the Batch Validation Message
- Calling the CZ_CF_API.VALIDATE Procedure
- Batch Validation Failure
- Skipping Batch Validation

Overview

This chapter describes using the runtime Oracle Configurator in programmatic mode, without direct end user interaction, which is called *batch validation*. This chapter includes information about:

> **Note:** Batch validation operates only on options that are BOM Model Items in Oracle Applications. Your host application must be part of Oracle Applications to implement batch validation.

Introduction

Batch validation allows a host application to perform tasks such as:

- Validating a BOM-based configuration in the background
- Determining a configuration quantity
- Deleting lines from a configured order while keeping the configuration valid
- Re-validating a previously booked order, if the configuration rules have changed in the meantime
- Using a custom user interface

A host application calls batch validation through the CZ_CF_API.VALIDATE PL/SQL procedure (see Calling the CZ_CF_API.VALIDATE Procedure, page 11-4). This procedure passes the batch validation message to the URL of the OC Servlet (see Passing the Batch Validation Message, page 11-2).

Passing the Batch Validation Message

A batch validation message consists of information defining the configuration context (such as an identifier for the configured model) and a list of configured options. The message can be used to revalidate a previously saved configuration.

The elements of the batch validation message are described in Elements of the Batch Validation Message, page 11-2.

An example of the batch validation message is provided in Example of Batch Validation Message, page 11-4.

The following table describes the elements of the batch validation message.

Elements of the Batch Validation Message

Element	Description
`<batch_validate>`	Composed of an `<initialize>` subelement, which initializes the configuration session, and a `<config_inputs>` subelement, which provides the inputs to the configuration (replacing the inputs provided by an interactive user). The `<batch_validate>` element can include the parameter `validation_type`, which indicates the type of validation to be performed.

Element	Description
validation_type	Optional parameter to the <batch_validate> element. Values are: • validate_order This value should be passed when validating orders, such as is done by Oracle Order Management. This is the default value. • validate_fulfillment This value should be passed when validating fulfillment status, such as is done by Oracle Install Base.Batch validation is never skipped when validation_type is validate_fulfillment. This value should not be passed if you want to skip batch validation. For more information see Skipping Batch Validation, page 11-9. • interactive This value should be passed if you need to conduct a batch validation session that behaves like an interactive end user configuration session. Example: <batch_validate validation_type="validate_order">
<initialize>	Described in Session Initialization, page 9-1. The parameters of the initialization message are described in Initialization Parameter Descriptions, page 9-17. See the description of the database_id, page 9-25 parameter for connectivity information.
<config_inputs>	Composed of a list of <option> elements.
<option>	Described in Session Termination, page 10-1. When an <option> element is used in a <config_inputs> element, only the <component_code> and <quantity> elements of the <option> are used.

Example of Batch Validation Message

```
<batch_validate validation_type="validate_order">
  <initialize>
    <param name="context_org_id">204</param>
    <param name="config_creation_date">03-25-2001-19-30-02</param>
    <param name="calling_application_id">300</param>
    <param name="responsibility_id">20559</param>
    <param name="config_header_id">21361</param>
    <param name="config_rev_nbr">1</param>
    <param name="read_only">FALSE</param>
    <param name="save_config_behavior">new_revision</param>
    <param name="database_id">ap115sun_dev115</param>
  </initialize>
  <config_inputs>
    <option>
      <component_code>143-1490-1494</component_code>
      <quantity>1</quantity>
    </option>
    <option>
      <component_code>143-297</component_code>
      <quantity>1</quantity>
    </option>
  </config_inputs>
</batch_validate>
```

Calling the CZ_CF_API.VALIDATE Procedure

If the host application is written in PL/SQL, it should call the VALIDATE procedure. CZ_CF_API.VALIDATE is the PL/SQL interface to batch validation. The VALIDATE procedure packages the inputs into a batch_validate init message and sends it to the configurator servlet. There are restrictions in the way that PL/SQL can request data from a URL that requires PL/SQL programs to use the CZ_CF_API.VALIDATE procedure, instead of passing the XML batch validation message.

For details on the parameters for CZ_CF_API.VALIDATE, see VALIDATE, page 17-69 in Programmatic Tools for Development, page 17-1.

Calling the CZ_CF_API.VALIDATE Procedure in a Program, page 11-4 shows fragments from a PL/SQL program that calls CZ_CF_API.VALIDATE.

Calling the CZ_CF_API.VALIDATE Procedure in a Script, page 11-6 shows a PL/SQL script that calls CZ_CF_API.VALIDATE.

Calling the CZ_CF_API.VALIDATE Procedure in a Program

```
...
```

```
/*-------------------------------------------------------------------------
---
Procedure Name : Send_input_XML
Description    : sends the xml batch validation message to hostapp that
has
                 options that are newly inserted/updated/deleted
                 from the model.
-------------------------------------------------------------------------
--*/
PROCEDURE Send_input_XML
            ( p_model_line_id       IN NUMBER ,
              p_org_id              IN NUMBER ,
              p_model_id            IN NUMBER ,
              p_config_header_id    IN NUMBER , 2003/10/20
              p_config_rev_nbr      IN NUMBER ,
              p_model_qty           IN NUMBER ,
              p_creation_date       IN DATE ,
              p_deleted_options_tbl IN   OE_Order_PUB.request_tbl_type
:= OE_Order_Pub.G_MISS_REQUEST_TBL,
              p_updated_options_tbl IN   OE_Order_PUB.request_tbl_type
:= OE_Order_Pub.G_MISS_REQUEST_TBL
              x_out_XML_msg         OUT NOCOPY LONG ,
              x_return_F            OUT NOCOPY VARCHAR2 )
...
      l_XML_hdr                       VARCHAR2(2000)
      l_html_pieces                   CZ_CF_API.CFG_OUTPUT_PIECES;
      l_option                        CZ_CF_API.INPUT_SELECTION;
      l_batch_val_tbl                 CZ_CF_API.CFG_INPUT_LIST;
      l_url                           VARCHAR2(500):=
FND_PROFILE.Value('CZ_UIMGR_URL');
      l_validation_type               CZ_API_PUB.VALIDATE_ORDER;
...
       Create_hdr_XML
            ( p_model_line_id        => p_model_line_id ,
              p_org_id               => p_org_id ,
              p_model_id             => p_model_id ,
              p_config_header_id     => p_config_header_id ,
              p_config_rev_nbr       => p_config_rev_nbr ,
              p_model_qty            => p_model_qty ,
              p_creation_date        => p_creation_date ,
              x_XML_hdr              => l_XML_hdr);
...
CZ_CF_API.Validate( config_input_list => l_batch_val_tbl ,
                         init_message      => l_XML_hdr ,
                         config_messages   => l_html_pieces ,
                         validation_status => l_validation_status ,
                         URL               => l_url
                         p_validation_type => l_validation_type );
```

Calling the CZ_CF_API.VALIDATE Procedure in a Script

```
set serveroutput on
set verify off
-- Run this query in SQL*Plus, providing input of model id
-- This query is like what the host application might send.
-- The output might go back to some other servlet.
BEGIN
declare
  config_input_list  CZ_CF_API.CFG_INPUT_LIST;
  ---- OC Servlet URL needs to be entered here....
  l_url varchar2(100):=
'http://www.mysite.com:10130/OA_HTML/configurator/UiServlet';
  init_message varchar2(4000):='<initialize>';
  config_messages  CZ_CF_API.CFG_OUTPUT_PIECES;
  validation_status  NUMBER;
  list_indx number := 1 ;
  l_validation_type CZ_API_PUB.VALIDATE_ORDER;
  begtime varchar2(30) := null ;
  endtime varchar2(30) := null ;
--- Build the initialization message.
        TYPE param_name_type IS TABLE OF VARCHAR2(25)
                        INDEX BY BINARY_INTEGER;
        TYPE param_value_type IS TABLE OF VARCHAR2(40)
                        INDEX BY BINARY_INTEGER;
        param_name      param_name_type;
        param_value     param_value_type;
        l_rec_index         BINARY_INTEGER;
        l_context_org_id                VARCHAR2(30);
        l_config_creation_date          VARCHAR2(30);
        l_two_task                      VARCHAR2(30);
        l_user                          VARCHAR2(30);
        l_pwd                           VARCHAR2(30);
        l_fndnam                        VARCHAR2(30);
        l_calling_application_id        VARCHAR2(30);
        l_responsibility_id             VARCHAR2(30);
        l_model_id                      VARCHAR2(30);
        l_config_header_id              VARCHAR2(30);
        l_config_rev_nbr                VARCHAR2(30);
        l_gwyuid                        VARCHAR2(30);
        l_read_only                     VARCHAR2(30);
        l_save_config_behavior          VARCHAR2(30);
        l_save_usage_behavior           VARCHAR2(30);
        l_ui_type                       VARCHAR2(30);
        l_so_line_id                    VARCHAR2(30);
        l_validation_org_id             VARCHAR2(30);
        l_dbc                           VARCHAR2(30);
        l_model_quantity                VARCHAR2(30);
        l_termination                   VARCHAR2(30);
        l_alt_database_name             VARCHAR2(40);
--Options
        l_component_code                VARCHAR2(2000);
        l_option_quantity               VARCHAR2(30);
        l_test_param                    VARCHAR2(20);
BEGIN
        param_name(1) := 'context_org_id';
        param_name(2) := 'config_creation_date';
        param_name(3) := 'two_task';
        param_name(4) := 'user';
        param_name(5) := 'pwd';
        param_name(6) := 'fndnam';
        param_name(7) := 'calling_application_id';
```

```
param_name(8) := 'responsibility_id';
        param_name(9) := 'model_id';
        param_name(10) := 'config_header_id';
        param_name(11) := 'config_rev_nbr';
        param_name(12) := 'gwyuid';
        param_name(13) := 'read_only';
        param_name(14) := 'save_config_behavior';
        param_name(15) := 'save_usage_behavior';
        param_name(16) := 'model_quantity';
        param_name(17) := 'database_id';
        param_name(18) := 'terminate_msg_behavior';
        param_name(19) := 'alt_database_name';
        SELECT
                '204',   -- corrected value
                '10-16-2000-09-41-12',
                null,
                null,
                null,
                null,
                '660',
                '50171',
                '143',  --this is the usual value for &modelId
                null,
                null,
                null,
                null,
                'new_revision',
                null,
                '45',
                'ap123dbs_dom123',
                'brief',
                'jdbc:oracle:thin:@serv01:1521:sid02'
        INTO
                l_context_org_id,
                l_config_creation_date,
                l_two_task,
                l_user,
                l_pwd,
                l_fndnam,
                l_calling_application_id,
                l_responsibility_id,
                l_model_id,
                l_config_header_id,
                l_config_rev_nbr,
                l_gwyuid,
                l_read_only,
                l_save_config_behavior,
                l_save_usage_behavior,
                l_model_quantity,
                l_dbc,
                l_termination,
                l_alt_database_name
        FROM    dual ;
        param_value(1) := l_context_org_id;
        param_value(2) := l_config_creation_date;
        param_value(3) := l_two_task;
        param_value(4) := l_user;
        param_value(5) := l_pwd;
        param_value(6) := l_fndnam;
        param_value(7) := l_calling_application_id;
        param_value(8) := l_responsibility_id;
```

```
param_value(9) := l_model_id;
        param_value(10) := l_config_header_id;
        param_value(11) := l_config_rev_nbr;
        param_value(12) := l_gwyuid;
        param_value(13) := l_read_only;
        param_value(14) := l_save_config_behavior;
        param_value(15) := l_save_usage_behavior;
        param_value(16) := l_model_quantity;
        param_value(17) := l_dbc;
        param_value(18) := l_termination;
        param_value(19) := l_alt_database_name;
        l_rec_index := 1;
        LOOP
                IF (param_value(l_rec_index) IS NOT NULL) THEN
                init_message  := init_message || '<param name=' ||
'"'    || param_name(l_rec_index) ||   '"'   ||'>'||
                                param_value(l_rec_index)||'</param>';
                END IF;
                EXIT WHEN l_rec_index > 18;  -- adjust for number of
parameters
                l_rec_index := l_rec_index + 1;
        END LOOP;
        init_message := init_message || '</initialize>';
        init_message := REPLACE(init_message,' ','+');
        dbms_output.enable(buffer_size => 200000);
                dbms_output.put_line(substr(init_message,1,255));
                dbms_output.put_line(substr(init_message,256,255));
                dbms_output.put_line(substr(init_message,512,255));
                dbms_output.put_line(substr(init_message,768,255));
                dbms_output.put_line(substr(init_message,1024,255));
                dbms_output.put_line(substr(init_message,1280,255));

CZ_CF_API.VALIDATE(config_input_list,init_message,config_messages,valida
tion_status,l_url,l_validation_type);
  IF(validation_status=CZ_CF_API.CONFIG_PROCESSED)THEN
    dbms_output.put_line('Config processed successfully');
  ELSIF(validation_status=CZ_CF_API.CONFIG_PROCESSED_NO_TERMINATE)THEN
    dbms_output.put_line('Config processed successfully, no termination
message');
  ELSIF(validation_status=CZ_CF_API.INIT_TOO_LONG)THEN
    dbms_output.put_line('Init message too long');
  ELSIF(validation_status=CZ_CF_API.INVALID_OPTION_REQUEST)THEN
    dbms_output.put_line('Invalid option request');
  ELSIF(validation_status=CZ_CF_API.CONFIG_EXCEPTION)THEN
    dbms_output.put_line('General config exception');
  ELSIF(validation_status=CZ_CF_API.DATABASE_ERROR)THEN
    dbms_output.put_line('Database error');
  ELSIF(validation_status=CZ_CF_API.UTL_HTTP_INIT_FAILED)THEN
    dbms_output.put_line('UTL_HTTP: initialization failed');
  ELSIF(validation_status=CZ_CF_API.UTL_HTTP_REQUEST_FAILED)THEN
    dbms_output.put_line('UTL_HTTP: request failed');
  ELSE
    dbms_output.put_line('Unknown error');
  END IF;
        l_rec_index := config_messages.FIRST;
        dbms_output.put_line ( 'Recieved Response from the server
follows ....' );
        LOOP
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),1,255))));
                dbms_output.put_line(
```

```
ltrim(rtrim(substr(config_messages(l_rec_index),256,255))));
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),512,255))));
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),768,255))));
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),1024,255))));
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),1280,255))));
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),1536,255))));
                dbms_output.put_line(
ltrim(rtrim(substr(config_messages(l_rec_index),1792))));
        EXIT WHEN l_rec_index = config_messages.LAST;
                l_rec_index := config_messages.NEXT(l_rec_index);
        END LOOP;
        dbms_output.put_line ('Servlet URL used follows ....');
    dbms_output.put_line(ltrim(rtrim(l_url)));
END;
END;
/
```

Batch Validation Failure

An end user can determine whether an order fails during batch validation if the imported order's quantities are not the same as the quantities in the original order, or if the quantities changed during an order cycle because the configuration model's rules have changed. For example, batch validation is run at booking time. If the published Model has changed from the initial order creation to booking time, then batch validation may result in different quantities causing the order to fail.

By setting the profile option CZ: Fail BV if Input Quantities Not Maintained, the end user can determine whether an order fails. This profile option is used in conjunction with the parameter in the Calling the CZ_CF_API.VALIDATE Procedure, page 11-4.

Batch Validation fails if the ordered configured BOM Items (input_list) do not match the batch validation BOM Items (from a previously processed configuration) and the profile optionCZ: Fail BV if Configuration Changed is set to Yes. If there is a difference between the ordered configured BOM Items and the batch validation BOM Items, then the differences are logged to CZ_CONFIG_MESSAGES.

For more information about the profile options, see the *Oracle Configurator Installation Guide*.

Skipping Batch Validation

A significant amount of batch validation processing time can be avoided when the CZ: Skip Validation Procedure profile option is set. If the profile option is set, then batch validate calls a customer created PL/SQL callback procedure. This callback procedure then makes the final decision based on the implementation requirements. For more information on the CZ: Skip Validation Procedure, see the *Oracle Configurator Installation Guide*.

The decision to skip batch validation is done on the batch server for each batch validation request. To skip parts of the batch validation process, the following criteria must be met:

- There are no input arguments.
- The skip profile option, CZ: Skip Validation Procedure is set to the name of the PL/SQL callback function. For more information see the *Oracle Configurator Installation Guide.*
- Effectivity date of the current configuration session is different from the effectivity date of the restored configuration and:
  - All nodes in the configuration model do not have effective start or end dates that are in the interval between the old and new effective dates.
  - All rules in the configuration model do not have effective start or end dates that are in the interval between the old and new effective dates.
- The publication record of the configuration that is being validated is the same as that of the saved configuration.
- The BOM Model quantity has not changed or is not provided in the initialization string
- The custom created PL/SQL callback function returns true

  When this function returns a value of `true`, the Batch Validation process does not perform all of its typical tasks, such as restoring the configuration and validating any inputs. A new configuration is saved when requested.
- The validation type is not `validate_fullfillment`. See Elements of the Batch Validation Message, page 11-2 for details.

PL/SQL Callback

A custom coded PL/SQL callback makes the final decision whether batch validation is skipped or not. A custom coded PL/SQL callback is needed because Configurator Extensions can change the configuration model. If there are no Configurator Extensions and you want to skip batch validation, then you must have a custom coded PL/SQL callback and enable the CZ: Skip Validation Procedure profile option. For more information on the CZ: Skip Validation Procedure, see the *Oracle Configurator Installation Guide*. Batch validation on its own cannot determine what a Configurator Extension does.

Specification of the PL/SQL Callback Function, page 11-11 shows the function's coding details:

Specification of the PL/SQL Callback Function

```
PROCEDURE my_skip_val_proc(
    p_root_inv_item_id IN NUMBER
    p_organization_id IN NUMBER
    p_config_creation_date IN DATE
    x_skip_validation OUT NOCOPY VARCHAR2
    x_return_status OUT NOCOPY VARCHAR2
    x_msg_data OUT NOCOPY VARCHAR2)
```

The PL/SQL callback arguments are described in the tablePL/SQL Callback Arguments, page 11-11:

PL/SQL Callback Arguments

Parameter	Data Type	Mode	Description
p_root_inv_item_id	number	in	Root BOM Model Inventory Item ID
p_organization_id	number	in	Root BOM Model Organization ID
p_config_creation_date	date	in	Configuration creation date
x_skip_validation	varchar2	out	Must return FND_API.G_TRUE if validation can be skipped; otherwise, return FND_API.G_FALSE
x_return_status	varchar2	out	Must return FND_API.G_RET_STS_SUCCESS if procedure completed successfully; otherwise return FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR if an error occurs within the procedure
x_msg_data	varchar2	out	Contains an error message if the procedure is returning an `x_return_status` value of FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR

PL/SQL Callback and Models that use Configurator Extensions

If you wish to skip batch validation and you have Models that use Configurator Extensions, then you must consider what the Configurator Extensions do when you write the callback function. If the Configurator Extension depends on the following, then the callback function should return a value of `false` and force validation to occur:

- Data held in custom tables that changes from time to time

- Data in Oracle Applications tables, other than the configuration model's definitions, that change from time to time. For example, MTL_SYSTEM_ITEMS flexfields.

- Data that is obtained by queries based on the CALLING_APPLICATION_HEADER_ID or CALLING_APPLICATION_LINE_ID that is provided in the Configurator initialization message. For example, SO_ORDER_HEADERS flexfield.

These dependencies could cause a Configurator Extension to make changes to the configuration and cause a validation failure.

12

Custom Integration

This chapter explains how to modify certain Oracle Configurator files as well as the purpose of the files and where they can be found.

This chapter covers the following topics:

- Overview
- General Directory Structure
- Files for the Servlet Directory
- Files for the HTML Directory
- Files for the Media Directory

Overview

To customize Oracle Configurator in your host application, you may need to modify certain Oracle Configurator files. This chapter describes:

As a prerequisite, you must have installed Oracle Configurator. See the *Oracle Configurator Installation Guide* for details.

You may wish to move certain files to other locations, to suit your site or host application requirements. This section describes constraints and guidelines on their location.

General Directory Structure

The table General Structure of Directories for Oracle Configurator, page 12-2 shows the directories required for the runtime Oracle Configurator, and their relationship. This general structure applies to all platforms, though the details may vary by platform. In some cases, the same physical directory may fill more than one role.

General Structure of Directories for Oracle Configurator

Directory Role	Description
OC Installation	The directory in which you install OC, based on your choice of installation directory in the Oracle Configurator setup program.
Servlet	Contains the Java class or archive files that implement the OC Servlet. Configurator Extensions and Return URL Servlets can be installed here. See the *Oracle Configurator Installation Guide* for more information.
Media	Contains the image files used by the runtime Oracle Configurator of your host application.
Log	Contains log files written by the runtime Oracle Configurator. See the *Oracle Configurator Installation Guide* for more information about logging.

Note that it is not necessary for the Servlet directory to have a separate physical location, because the files it contains are referenced by environment variables that you set while installing the runtime Oracle Configurator servlet.

Files for the Servlet Directory

The Servlet directory contains files that must be referenced in the PATH and CLASSPATH environment variables.

The table Files for the Servlet Directory, page 12-3 shows the files that should be installed in the Servlet directory.

Files for the Servlet Directory

File	For Platform	Comment
`libczlce.so`	Unix	Must be in the LD_LIBRARY_PATH environment variable parameter for your servlet.
`czlce.dll`	Windows NT	Must be in the PATH system environment variable on the host computer on which the servlet is installed. This should be set by the OC installation program.

Files for the HTML Directory

By default, the HTML directory is the directory pointed to by the Oracle Applications alias OA_HTML.

Files for the Media Directory

By default, the Media directory is the directory pointed to by the Oracle Applications alias OA_MEDIA.

The image files in the Media directory are used by the runtime Oracle Configurator to decorate your customized user interfaces, and also to represent application logic state in DHTML legacy user interfaces.

Important: As of this release, DHTML UIs are no longer supported.

These files must be compatible with web browser technology. You cannot use BMP (Windows bitmap) files in your user interface for the Oracle Configurator window, because this file format is not compatible with Web browsers. The runtime Oracle Configurator window can use GIF, JPG, and other formats compatible with Web browsers.

13

Pricing and ATP in Oracle Configurator

This chapter provides an overview of how pricing works in a runtime Oracle Configurator.

This chapter covers the following topics:

- Overview
- Introduction
- Runtime Oracle Configurator Pricing Architecture
- Runtime Pricing Behavior
- Integration of Pricing and ATP with Oracle Configurator
- Controlling Pricing and ATP in a Runtime Oracle Configurator

Overview

This chapter describes the integration of pricing and ATP with Oracle Configurator. It includes:

- Runtime Oracle Configurator Pricing Architecture, page 13-2
- Runtime Pricing Behavior, page 13-9
- Integration of Pricing and ATP with Oracle Configurator, page 13-10
- Controlling Pricing and ATP in a Runtime Oracle Configurator, page 13-12

> **Note:** If your host application is part of Oracle Applications, then the integration with pricing and ATP is already defined. You only need to implement pricing and ATP for custom host applications. The CZ_PRICING_STRUCTURES and CZ_ATP_REQUESTS tables must be populated for custom host applications to integrate with pricing and

ATP.

Introduction

How Oracle Configurator handles pricing and ATP (Available To Promise) data depends on the type of runtime Oracle Configurator you choose to use. A runtime Oracle Configurator can be called from a variety of different applications and requires an interface between the runtime Oracle Configurator and the host application's pricing mechanism. For more information on advanced pricing, see *Oracle Advanced Pricing User's Guide*.

Runtime Oracle Configurator Pricing Architecture

When the host application is part of Oracle Applications, such as Order Management, pricing data comes from Oracle Advanced Pricing (QP). The QP interface is highly configurable. Depending on how it is configured, it may be necessary that appropriate data records are defined in the host application to determine pricing parameters. The host application must implement the Oracle Configurator pricing interface package, as described in Pricing Callback Interface, page 13-4. Likewise, when the host application is not an Oracle Applications product, it must implement the Oracle Configurator pricing interface package, so that the runtime Oracle Configurator knows how to determine prices.

Therefore, the host application must provide an interface PL/SQL package that interacts whenever pricing is requested between the runtime Oracle Configurator and the host application's pricing engine. The runtime Oracle Configurator is displayed when the user clicks the Configure button in the host application. The runtime Oracle Configurator calls the pricing interface package to get:

- List prices for all selectable options in the configuration
- Selling prices for all selectable options in the configuration
- Total price for the entire configuration

The browser presents *either* list prices for all selectable options, *or* selling prices for all selected options, and enables you to add a total price.

For more information about the Pricing Callback Interface, see Pricing Callback Interface , page 13-4.

Pricing Callback Interface Package

The host application sends an initialization message to the runtime Oracle Configurator with the interface package and procedure name. The runtime Oracle Configurator calls

this interface package to get current pricing information for a single item or a list of items.

The interface package determines the full context in which to call the target pricing engine. The interface package then calls the pricing engine and captures all of the results, storing these results in tables (or some other Oracle session-insensitive place) for future reference when the runtime Oracle Configurator session exits. The runtime Oracle Configurator does not reference the contents of these tables.

The interface package temporarily writes the list and/or selling prices for the configuration components in the temporary CZ_PRICING_STRUCTURES table so that they can be presented to the end user.

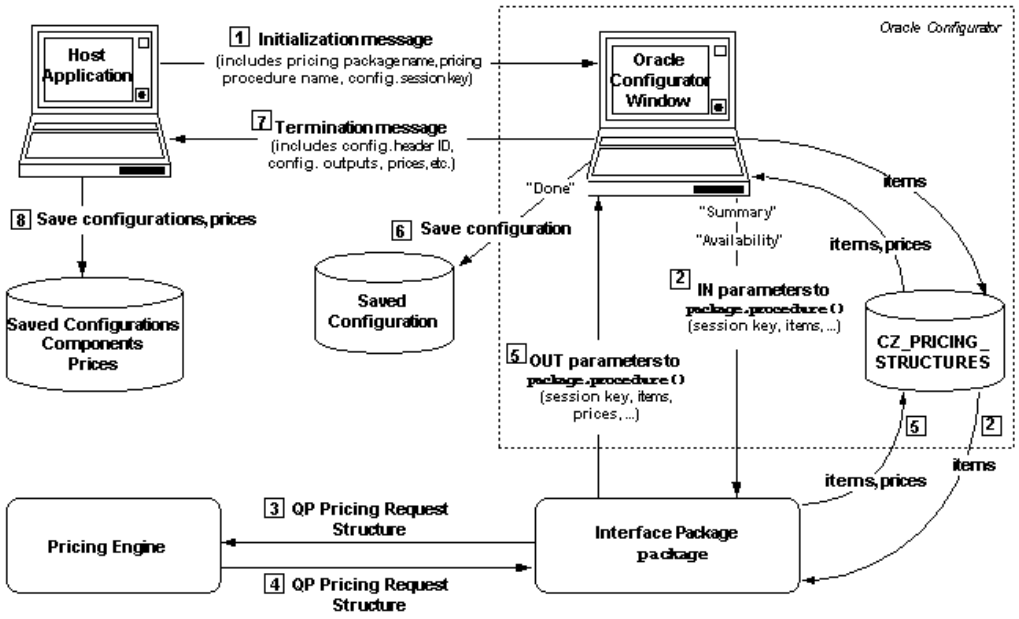
The CZ_PRICING_STRUCTURES table does not support pricing rules based on the fact that items belong to the same instance. Pricing is done per component instance.

The runtime Oracle Configurator saves the configuration information in the appropriate CZ tables. The runtime Oracle Configurator does *not* save list or selling prices. It is up to the host application to save configuration data, list prices, and selling prices in its own tables. For example, Order Management stores the configuration in OE_ORDER_LINES_ALL, and stores the pricing data in OE_PRICE_ADJUSTMENTS. The host application decides whether it is necessary to recalculate prices depending on the value of the `prices_calculated_flag` in the runtime Oracle Configurator termination message.

When the host application calls the runtime Oracle Configurator to edit an existing configuration, the runtime Oracle Configurator asks the interface package for the current list and selling prices of the currently selected components.

Runtime Oracle Configurator Pricing Architecture, page 13-4, illustrates this architecture. Illustrated steps 2 through 5 can be repeated many times. Note that in Runtime Oracle Configurator Pricing Architecture, page 13-4, all of the database symbols refer to the same instance of the CZ schema.

Runtime Oracle Configurator Pricing Architecture



See the Pricing Callback Interface, page 13-4 for details about the pricing interface package, and see Session Initialization, page 9-1 and Session Termination, page 10-1 for details about the initialization and termination messages for a runtime Oracle Configurator session.

Pricing Callback Interface

The pricing callback interface package provides interfaces for these distinct procedures:

- Price Multiple Items
- Price Multiple Items for MLS

The Price Multiple Items procedure returns price information for a group of items. The Price Multiple Items Procedure Parameters, page 13-4 table describes the parameters for this procedure.

Price Multiple Items Procedure Parameters

Parameter	In/Out	Type	Required	Note
`configurator_session_key`	In	Varchar2	Required	Limit of 50 characters

Parameter	In/Out	Type	Required	Note
price_type	In	Varchar2	Required	Values are: LIST, SELLING, or BOTH
config_total_price	Out	Number nocopy	n/a	

The Price Multiple Items MLS procedure returns price information for a group of items. The Price Multiple Items MLS Procedure Parameters, page 13-5 table describes the parameters for this procedure.

Price Multiple Items MLS Procedure Parameters

Parameter	In/Out	Type	Required	Note
configurator_session_key	In	Varchar2(50)	Required	Limit of 50 characters
price_type	In	Varchar2	Required	Values are: LIST, SELLING or BOTH
config_total_price	Out	Number nocopy	n/a	
currency_code	Out	Varchar2 nocopy	n/a	

The parameters of the interface are passed by positional notation, so you can name the parameters as wanted, as long as you retain the positionality specified in Price Multiple Items Procedure Parameters, page 13-4 and Price Multiple Items MLS Procedure Parameters, page 13-5.

Use of the Database in the Price Multiple Items Procedures

When you specify the Price Multiple Items procedures, Oracle Configurator stores the list of items to be priced in the database table CZ_PRICING_STRUCTURES. This columns in this table are described in CZ_PRICING_STRUCTURES Interface Table, page 13-6.

CZ_PRICING_STRUCTURES Interface Table

Column Name	Data Type	Null?	Description
CONFIGURATOR_SESSION_KEY	Varchar2	Not Null	Limit of 50 characters. Primary key. Identifies a configurator session. Only one configuration can be handled in the session.
SEQ_NBR	Number	Not Null	Primary key. Sequence number of the item in the list of items.
PS_NODE_ID	Number		Limit of 9 digits. PS_NODE_ID is a foreign key reference into the CZ_PS_NODES table, which defines the "configuration" identity of the object.
ITEM_KEY	Varchar2	Not Null	Limit of 2000 characters. ORIG_SYS_REF for imported items or PS_NODE_ID for non-imported items.
ITEM_KEY_TYPE	Number	Not Null	Limit of 9 digits. Set to 1Value of CZ_PRC_CALLBACK_UTIL.G_ITEM_KEY_BOM_NODE. if ITEM_KEY is ORIG_SYS_REF. Set to 2Value of CZ_PRC_CALLBACK_UTIL.G_ITEM_KEY_PS_NODE. if ITEM_KEY is PS_NODE_ID.
QUANTITY	Number		Limit of 9 digits. Item quantity
UOM_CODE	Varchar2		Limit of 3 characters. UOM code
LIST_PRICE	Number		List price
SELLING_PRICE	Number		Selling price

Column Name	Data Type	Null?	Description
MSG_DATA	Varchar2		Limit of 2000 characters. Message text filled in by your host application.
CONFIG_ITEM_ID	Number	Not Null	This corresponds to the CZ_CONFIG_ITEMS.CONFIG_ITEM_ID. Note: CZ_PRICING_STRUCTURES.ITEM_KEY is unable to establish the full hierarchy of a configuration when there are multiple instantiations.
PARENT_CONFIG_ITEM_ID	Number		Together with CONFIG_ITEM_ID, this establishes the full hierarchy of the configuration when there are multiple instantiations.

Your pricing package must retrieve the items from this table and call the pricing engine, then capture all of the results and update the CZ_PRICING_STRUCTURES table with list and/or selling prices, and any message text. Oracle Configurator retrieves the prices from the CZ_PRICING_STRUCTURES table during the configuration session, so that they can be presented in the Oracle Configurator window. When the Oracle Configurator window exits, Oracle Configurator deletes the pricing records from the CZ_PRICING_STRUCTURES table.

If your host application must retain the prices for use after the end of the current configuration session, then your pricing package must store the results in application-specific tables (or some other location that is insensitive to the Oracle session). Oracle Configurator does not reference the contents of these application-specific tables.

Examples of the Pricing Callback Interface

Pricing Callback Interfaces must populate the CZ_PRICING_STRUCTURES table.

Pricing Callback Interface, page 13-7 shows a possible implementation of the callback interface for multiple-item pricing procedures.

Initialization Message Using Release 12Pricing and ATP Parameters, page 13-12 shows how to specify pricing parameters in your initialization message.

Pricing Callback Interface

```
PACKAGE CZ_PRICE_TEST AUTHID CURRENT_USER AS
PROCEDURE price_multiple_items (p_configurator_session_key IN VARCHAR2,
                                p_price_type IN VARCHAR2,
                                p_total_price OUT NUMBER);
END;
```

ATP Callback Interface

The "Get ATP Dates" procedure returns availability dates for all PTO Models but only returns the date for the **ATO** top level Model. The table ATP Procedure Parameters, page 13-8 describes the parameters for the Get ATP Dates procedure.

ATP Procedure Parameters

Parameter	In/Out	Type	Required	Note
configurator_session_key	In	Varchar2	Required	Limit of 50 characters
warehouse_id	In	Number	Required	
ship_to_org_id	In	Number	Conditionally Required	You must provide either `ship_to_org_id` (by itself), or both `customer_id` and `customer_site_id`.
customer_id	In	Number	Conditionally Required	You must provide either `ship_to_org_id` (by itself), or both `customer_id` and `customer_site_id`.
customer_site_id	In	Number	Conditionally Required	You must provide either `ship_to_org_id` (by itself), or both `customer_id` and `customer_site_id`.
requested_date	In	Date	n/a	If a date is not provided, then the date defaults to the `SYSDATE`.
ship_to_group_date	Out	Date nocopy	n/a	

The parameters of the interface are passed by positional notation, so you can name the parameters whatever you want, as long as you retain the positionality specified in ATP Procedure Parameters, page 13-8.

Use of the Database with the ATP Callback Interface

When you specify the Get ATP Dates procedure, Oracle Configurator stores the list of items to obtain ATP dates for in the database table CZ_ATP_REQUESTS. For details on Oracle Configurator tables, see the Oracle Integration Repository.

If you are using the Oracle ATP pricing mechanism, then your ATP package must retrieve the items from the table and call the `call_atp()` procedure defined in your ATP package, then capture all of the results and update the CZ_ATP_REQUESTS table with ATP dates.

Oracle Configurator retrieves the ATP dates from the CZ_ATP_REQUESTS table during the configuration session, so that they can be presented in the Oracle Configurator window. When the Oracle Configurator window exits, OC deletes the ATP dates from the CZ_ATP_REQUESTS table.

If your host application must retain the ATP dates for use after the end of the current configuration session, then your ATP package must store the results in application-specific tables (or some other location that is insensitive to the Oracle session). Oracle Configurator does not reference the contents of these application-specific tables.

Examples of the ATP Callback Interface

ATP Callback Interface, page 13-9 shows an implementation of the callback interface for ATP procedures.

Initialization Message Using Release 12Pricing and ATP Parameters, page 13-12 shows how you would specify ATP parameters in your initialization message.

Example of Callback ATP Procedure, page E-3 provides an example in context.

ATP Callback Interface

```
PACKAGE cz_atp_callback AS
  PROCEDURE call_atp (p_config_session_key IN VARCHAR2,
                      p_warehouse_id IN NUMBER,
                      p_ship_to_org_id IN NUMBER,
                      p_customer_id IN NUMBER,
                      p_customer_site_id IN NUMBER,
                      p_requested_date IN DATE,
                      p_ship_to_group_date OUT NOCOPY DATE);
END cz_atp_callback;
```

Runtime Pricing Behavior

It is important to understand some aspects of pricing behavior in the runtime Oracle Configurator, as they can affect both performance and the responsibilities of the host application.

- The runtime Oracle Configurator caches list prices of the items until it is terminated. The runtime Oracle Configurator assumes that the list price of any item does not depend on which other items are selected and remains unchanged during

the configuration session.

- The runtime Oracle Configurator's performance depends critically on the performance of the pricing interface package that you provide. List prices in particular must be returned very quickly, because they are demanded for every option that is displayed.

- The runtime Oracle Configurator does not save computed prices. If, after the configuration session ends, the host application requires access to prices that were computed during the session, it is up to the host application's interface package to save the computed prices. Prices should be saved together with enough information to allow them to be correlated with the components of the saved configuration.

- If the runtime Oracle Configurator is initialized with a previously saved configuration, it is up to the host application to either return the saved list and selling prices or to call the pricing engine to get the current price. Direct or manual editing of prices, adjustments, discounts, and so on is the responsibility of the host application.

Integration of Pricing and ATP with Oracle Configurator

Integrating the Oracle Configurator window with your pricing or ATP implementation consists primarily of causing your host application to post the XML initialization message to the OC Servlet (for example, through the coding of the Configure button), passing as initialization parameters the names of your packages and procedures.

To use the OC pricing and ATP interfaces, you must:

1. Install the OC interface packages in your database, by installing Oracle Configurator with Oracle Rapid Install. See Database Compatibility, page 13-11.

2. Write your own PL/SQL pricing or ATP procedures, using the OC interfaces. See Pricing and ATP Callback Procedures, page E-2 for examples.

3. Install your packages containing your procedures into the Oracle Applications database.

   You can interface to the Oracle QP pricing engine from your own procedures.

4. In the initialization message that your host application passes to the OC Servlet, provide parameters that specify the name of the pricing package, the name of the ATP package, the procedure to use, and the type of pricing to perform.

   See Initialization Parameters, page 13-11 for an example. See Pricing Parameters, page 9-15 and ATP Parameters, page 9-15 for explanation of the parameters.

   You can test the effect of pricing and ATP parameters when you test your Model in Oracle Configurator Developer, by entering the Pricing Package and ATP Package

parameters in the in the Custom Initialization Parameters field of the Test Preferences page.

5. Enable pricing display and update behavior, as described in Controlling Pricing and ATP in a Runtime Oracle Configurator, page 13-12.

   Note: The display and updating of pricing are controlled by the values of the database fields CZ_UI_DEFS.PRICE_DISPLAY and CZ_UI_DEFS.PRICE_UPDATE. If these fields are null, then the information is not displayed. For details on these tables, see the Oracle Integration Repository.

Database Compatibility

Oracle Configurator works with Oracle Applications Release 12. To determine the database version supported by Oracle Applications, refer to the Certify and Availability on the Oracle Support Web site.

To obtain pricing data from an Oracle Enterprise Edition database, as used with Oracle Applications 10.7, 11.0, you must run a concurrent program. See the Populate and Refresh Configuration Models Concurrent Programs, page C-18.

There are several likely scenarios for pricing and ATP integration. These scenarios are described in the following table:

To Integrate with...	Do the following ...
Oracle Applications Release 12 database	Write your own callback procedures (which can call the QP Advanced Pricing engine). To import BOM Model data to the CZ schema tables, you run concurrent programs in the Oracle **Bills Of Material** application. To export orders to Order Management (Oracle Applications Release 12), you use existing or new programming in your -host application.
Third-party database	For both import and export of pricing data, you must write custom programs.

You can use the callback interface in all these scenarios.

Initialization Parameters

Initialization Message Using Release 12 Pricing and ATP Parameters, page 13-12 is a test page that shows how you would specify pricing and ATP parameters in your initialization message. The names of the pricing and ATP parameters are typographically emphasized. This example shows parameters for use with Oracle

Applications Release 12. See Pricing Parameters, page 9-15 and ATP Parameters, page 9-15.

Initialization Message Using Release 12 Pricing and ATP Parameters

```
<html>
<head>
<title>Pricing Test</title>
</head>
<script language="javascript" >function init()
{document.test1.submit();}</script>
<body onload="init();">
<form action="http://www.mysite.com:8802/OA_HTML/CZInitialize.jsp"
method="post" id="test1" name="test1"><input type="hidden" name="XMLmsg"
value='<initialize>
<param name="database_id">serv02_sid01</param>
<param name="user">operations</param>
<param name="pwd">welcome</param>
<param name="calling_application_id">708</param>
<param name="responsibility_id">22713</param>
<param name="ui_type">JRAD</param>
<param name="ui_def_id">3080</param>
<param name="pricing_package_name">cz_price_test</param>
<param name="price_mult_items_proc">price_multiple_items</param>
<param name="configurator_session_key">1234</param>
<param name="atp_package_name">cz_atp_callback_stub</param>
<param name="get_atp_dates_proc">call_atp</param>
<param name="warehouse_id">207</param>
<param name="customer_id">1000</param>
<param name="customer_site_id">1567</param>
</initialize>'>
</form>
<br>Loading ...
</body>
</html>
```

To obtain the final prices calculated by your pricing package and ATP package, you need to specify a value of `full` for the initialization parameter terminate_msg_behavior, page 9-35. When your configuration session terminates normally, Oracle Configurator returns the final prices in the termination message. Your host application can then save the prices as needed.

Controlling Pricing and ATP in a Runtime Oracle Configurator

This section describes how to display prices and Available to Promise (ATP) information in a runtime Oracle Configurator.

Following is an overview of the process:

1. Define the following profile options, as required: CZ: Enable List Prices, CZ: Enable Selling Prices, CZ: Enable ATP.

   For details, see the *Oracle Configurator Installation Guide*.

2. In Oracle Configurator Developer, select pricing and ATP settings for the generated User Interface.

For details, see Displaying Prices and ATP Information, page 13-13.

3. If you are deploying a custom application, set the appropriate parameters in the initialization message that is posted to the OC Servlet.

   For details about the initialization and termination messages for pricing and ATP, see Session Initialization, page 9-1 and Session Termination, page 10-1.

   For details about the pricing interface package, see Pricing Callback Interface , page 13-4.

Displaying Prices and ATP Information

If you have defined the required profile options, you can control which types of prices and availability information is displayed and how they are updated in a generated User Interface. To do this, edit the UI Definition in Oracle Configurator Developer and modify the Price and Availability Display settings.

For example, you set CZ: Enable List Prices and CZ: Enable ATP to Yes. You can prevent list prices and ATP data from appearing in a UI by deselecting the List Prices and Availability settings in the UI Definition.

For details about the pricing and ATP settings available at the UI Definition level, and how to modify them, see the *Oracle Configurator Developer User's Guide*.

Updating Prices

If pricing is enabled and the UI Definition's pricing settings are set to display prices at runtime, the Recalculate Prices setting controls what action causes selling prices to be updated. You can set this to

- On Request
- On Page Load
- On Change

For details about these settings, see the *Oracle Configurator Developer User's Guide*.

Examples of Controlling Pricing

This section lists how the various settings that control pricing can be used together.

Example: List Prices Only

List Price Profile Option and UI Definition Settings, page 13-14 lists the recommended property or setting if you want to display only list prices at runtime.

List Price Profile Option and UI Definition Settings

Profile Option or Setting	Value
`CZ: Enable List Prices`	`Yes`
Price Display Style	List Price
Price Update	On Request

Example: Selling Prices Only

The table Selling Price Profile Option and UI Definition Settings, page 13-14 lists recommended settings if you want to display only selling prices.

Selling Price Profile Option and UI Definition Settings

Property or Setting	Value
`CZ: Enable Selling Prices`	`Yes`
Price Display Style	Selling Price
Price Update	On Request

14

Multiple Language Support

This chapter explains how Item descriptions are entered in Oracle Applications and can be displayed in multiple languages when deploying an Oracle Configurator User Interface.

This chapter covers the following topics:

- Overview
- Introduction
- Data Import
- Installed Languages in Multiple Server Environments
- Deploying a User Interface that Supports MLS
- Translating Data in CZ_LOCALIZED_TEXTS
- Translating XML Documents

Overview

This chapter describes the impact of Multiple Language Support (MLS). It includes:

For general information about creating a configuration model and User Interface that can be deployed in multiple languages, see the *Oracle Configurator Developer User's Guide*.

For additional information about MLS, refer to the following sources:

- *Oracle E-Business Suite Concepts*: This document contains general information about language support in Oracle Applications.
- *Oracle E-Business Suite Installation Guide: Using Rapid Install*: The chapter on setting up National Language Support contains a list of languages supported by all Oracle Applications products.

Introduction

All predefined Configurator Developer messages are stored in the following tables:

- FND_NEW_MESSAGES
- FND_LOOKUPS
- CZ_LOOKUP_VALUES_VL

Oracle translates all messages in this table into each installed language.

All text that a Configurator Developer user enters that appears in a generated UI is stored in the CZ_LOCALIZED_TEXTS table in the user's base language. For a list of all Configurator Developer text that is stored in this table, see the *Oracle Configurator Developer User's Guide*. If you are deploying a configuration model and UI in other languages, then the data in this table must be translated.

Translating text into different languages is typically accomplished by:

- Extracting the database file (text) into a legible and editable format by spooling the output of a query from SQL*Plus
- Sending the file to a third-party company that edits the file and translates the data
- Re-uploading the file to the database using SQLLoader

This process is described in Translating Data in CZ_LOCALIZED_TEXTS, page 14-4.

Data Import

Before importing a BOM Model, be sure that all Items defined in Oracle Inventory have descriptions. All translated Item descriptions are stored in the MTL_SYSTEM_ITEMS_TL table.

The Populate Configuration Models concurrent program:

- Extracts all strings associated with BOM Models imported from MTL_SYSTEM_ITEMS_TL for all languages installed on the import target database

- Populates CZ_LOCALIZED_TEXTS with MTL_SYSTEM_ITEMS_TL.DESCRIPTION

New Models

When importing a new BOM Model, the Oracle Configurator import procedures import all translated descriptions of each BOM Model item.

Existing Models

When refreshing an existing imported BOM Model, the import procedures update the CZ_LOCALIZED_TEXTS table if translations were added or modified in Oracle Inventory.

For more information, see Refreshing Imported Data, page 5-16.

Installed Languages in Multiple Server Environments

Publishing in a multi-server environment (such as multiple development instances and a production instance), requires that source and target instances have the same base language as well as the same set of installed languages. If either the base language or the set of installed languages are not the same, then the concurrent program fails when copying the publication to the target database or when migrating Models from one development database to another. When the source and target instances have the same base language and set of installed languages, then any missing or superfluous data in the target database (which can cause errors at runtime) is eliminated. For more information, see Database Instances, page 3-1.

Deploying a User Interface that Supports MLS

Like Configurator Developer, all Oracle Applications products that can host an Oracle Configurator use the Languages setting to control the session language. For more information on the Languages setting, see the *Oracle Configurator Developer User's Guide*. When a host application launches Oracle Configurator to configure an item, the language specified in the database ICX session ticket is passed to Oracle Configurator. Oracle Configurator uses this information to determine which translated text to retrieve from the database and display in the UI.

> **Note:** When a new language is added in Oracle Applications and you want to see the user interface labels in the new language, you must re-publish the Models.

For more information about deploying a UI, see User Interface Deployment, page 19-1 .

Translating Data in CZ_LOCALIZED_TEXTS

Following is an example of how you can extract and translate data in CZ_LOCALIZED_TEXTS.

1. Extract data from CZ_LOCALIZED_TEXTS using SQL*Plus.

   For example:

```
SQL> set linesize 2000
set heading off
spool <file>
select
to_char (intl_text_id) ||
',"' ||
 to_char (model_id) || ',' ||
 to_char (ui_def_id) || ',' ||
 language || '","' ||
 source_lang || '","' ||
 replace (localized_str, '"', '""') || '"'
 from
 cz_localized_texts
 where
 language = 'US' and
 deleted_flag = '0' and
 (
 model_id in (4687, 8546, 11574) or
 ui_def_id in (68487, 56468, 8375)
 )
 ;
 spool off
```

   > **Note:** The query in this example extracts only the 'US' records (language = 'US'). If you need to translate the text into multiple languages, copy the file for each target language. Alternatively, you can extract all translations by removing this filter in the query.

2. Edit the file and translate the text. (This is typically performed by a third party that specializes in translating data.)

   For example:

```
SQL> 78546,4687,68487,"US","US","Here ""Harry"" is a dog"
92115,4687,68487,"FR","FR","Ici <<Henri>> est chien"
```

   Note that all string data is in quotation marks. Quotation marks within the translatable strings are doubled but they may need to be altered to fit quotation conventions in the target language. The LANGUAGE and SOURCE_LANG values should be changed to the target language of the translation.

3. Delete the existing records.

   For example:

```
SQL> delete from cz_localized_texts
 where
 (
model_id in (4687, 8546, 11574) or
 ui_def_id in (68487, 56468, 8375)
 )
```

In this example, the script does not contain the filters "deleted_flag = '0' and language = 'US' " because it removes the deleted records and replaces them with the new translations.

4. Load the data using SQLLoader.

   For example:

```
SQL> sqlldr userid=apps@CUSTDB
 control=loadtexts.ctl
 log=loadtexts.log
```

   Below is an example of an SQLLoader control file:

```
LOAD DATA
 INFILE 'customer_texts.dat'
 BADFILE 'customer_texts.bad'
APPEND
 INTO TABLE CZ.CZ_LOCALIZED_TEXTS
 FIELDS TERMINATED BY ","
 OPTIONALLY ENCLOSED BY '"'
 (INTL_TEXT_ID, MODEL_ID,
 UI_DEF_ID, LANGUAGE,
 SOURCE_LANG, LOCALIZED_STR)
```

5. Translate XML documents as necessary.

   See Translating XML Documents, page 14-5.

Translating XML Documents

After translating all text in CZ_LOCALIZED_TEXTS and unit testing the UI, it is possible that some text in your UI pages (XML documents) will still require translation. Some examples include the text of a Static Styled Text UI element and column header text for elements that represent the columns of a table. For details about these UI elements, see the *Oracle Configurator Developer User's Guide*.

The Oracle Applications Extension Translation toolset deals with translatable information contained in OA Extension pages using XLIFF, a widely used XML format for transferring and manipulating translatable resources. You can use this toolset to translate the XML documents that make up your generated UI.

For details, refer to the following documents, which are available on Oracle Applications Documentation, on the Oracle Technology Network:

- *Oracle Application Framework Personalization Guide*

- *Oracle Application Framework Developer's Guide*

Part 4

Configuration Model

This Part presents information that enables you to extend a BOM Model's structure, rules, and UI to reflect your business requirements and integrate with a host application as described in Model Development Tasks, page 1-6.

15

Controlling the Development Environment

This chapter covers the following topics:

- Overview
- Setting up Oracle Configurator Developer
- Setting up Access to Configurator Developer
- Oracle Configurator Developer

Overview

This chapter presents the following topics:

Setting up Oracle Configurator Developer

To utilize some Oracle Configurator Developer functionality or access a runtime Oracle Configurator from other Oracle Applications such as Order Management, you must set some profile options. See the *Oracle Configurator Installation Guide* for information about Oracle Configurator Developer profile options.

Multiple Language Support (MLS) enables you to create a Model and one or more user interfaces in your base language and then display the runtime UI in any language in which you do business. For more information on MLS see the *Oracle Configurator Developer User's Guide* and Multiple Language Support, page 14-1.

For background on the relationship of Oracle Configurator Developer to the Oracle Configurator architecture, see Configurator Architecture, page 2-1.

Setting up Access to Configurator Developer

Some setup is required to provide access to Configurator Developer. This section provides an overview of the process.

Access to specific Configurator Developer functions, such as creating Model structure, defining rules, and generating a User Interface, is controlled by the responsibility to which each Oracle Applications user is assigned. For example, a responsibility may enable user CTHOMAS to generate UIs, but not allow that user to define or modify rules.

For more information about Oracle Applications responsibilities and function security, see the *Oracle E-Business Suite System Administrator's Guide*.

To set up access to Oracle Configurator Developer, your System Administrator must:

1. Define Oracle Configurator Developer users in Oracle Applications.

   For details, see the *Oracle E-Business Suite System Administrator's Guide*.

2. Assign at least one of the predefined Configurator Developer responsibilities listed below in *The Predefined Configurator Developer Responsibilities* to each Oracle Configurator Developer user.

p

The following table describes the predefined Oracle Configurator Developer responsibilities.

The Predefined Configurator Developer Responsibilities

Responsibility	Description
Configurator Administrator	Access to *all* forms-based Oracle Configurator related concurrent programs. For more information on concurrent programs, see Concurrent Programs, page C-1.
Configurator Developer	Access to *some* forms-based Oracle Configurator related concurrent programs. For more information on concurrent programs, see Concurrent Programs, page C-1.

Responsibility	Description
Oracle Configurator Administrator	Can create, edit, and delete the same objects as the Configurator Developer responsibility (see below). Can create, import, migrate, refresh, publish, synchronize, and populate Models. Has access to *all* HTML-based Oracle Configurator related concurrent programs. For more information on concurrent programs, see Configurator Administration Concurrent Programs, page C-2 for more information.
Oracle Configurator Developer	Unrestricted read-only access to all objects (including Model structure, rules, User Interfaces, UI Templates, and so on). Can create, edit, and delete the following: Folders, Model structure; rules and rule folders; Properties; Items and Item Types; Usages; Effectivity Sets; UI Templates; User Interfaces. Can create, import, refresh, publish, and populate Models. Has access to some HTML-based Oracle Configurator related concurrent programs. For more information on concurrent programs, see Concurrent Programs, page C-1.
Oracle Configurator Viewer	Unrestricted read-only access to all objects (including Model structure, rules, User Interfaces, UI Templates, and so on). Cannot modify any objects.

Warning: Oracle strongly recommends that you do *not* modify the predefined Oracle Configurator Developer responsibilities. If you need to provide access to a different combination of menus and functions, then define new responsibilities in Oracle Applications. For information about defining responsibilities, see the *Oracle E-Business Suite System Administrator's Guide*.

Oracle Configurator Developer

Oracle Configurator Developer provides an intuitive and powerful environment for the creation and maintenance of configuration models.

Model Development

Using Oracle Configurator Developer, the developer makes modifications to a Model

(structure, rules, UI definitions). These modifications of the model data are committed to the Oracle Applications database server. This is shown as the Model development environment in Three tier Architectural Overview of Oracle Configurator Developer, page 2-12.

After making modifications to the Model, the Model can be tested in either the runtime Oracle Configurator or the Model Debugger. For more information see the *Oracle Configurator Developer User's Guide*.

The unit-testing configuration data is committed to the database, after the developer clicks Apply or Finish.

Runtime Testing

All Oracle Configurator runtime database commits are through JDBC. When the end user closed the Configurator window, the resulting configuration data is saved directly to the database.

To test the configuration model, there are certain objects that must be in place:

- In order for Functional Companions to run, you must have access to Java classes. For more information, see the *Oracle Configurator Developer User's Guide*.
- OC Servlet must be restarted if you add or modify the Java class for a Configurator Extension.
- Open a new configuration session in a new browser window by going to the Model's Utility page to view any Model, rules, or UI changes.
- Check that the OC Servlet is running and what version of the runtime Oracle Configurator software is being used. Enter the following URL in a browser using the specific local settings for host and port where the OC Servlet is installed:

  Example

  ```
  http://host:port/OA_HTML/configurator/UiServlet?test=version
  ```

For more information about the runtime Oracle Configurator and Oracle Configurator Developer, see Configurator Architecture, page 2-1.

16

Publishing Configuration Models

This chapter explains the database processes for publishing configuration models to make them available to host applications.

This chapter covers the following topics:

- Overview
- Planning Publications
- How Host Applications Select a Published Model
- Defining a Publication
- Publishing a Configuration Model
- Maintaining Publications

Overview

This chapter presents information about:

Planning Publications

Publishing is a process that creates a copy of a configuration model on a specific database and makes it available to host applications for testing or production use. The copied data is called a **publication**, and it includes the Model's structure, rules, User

Interface, and Global User Interface Templates. The publishing process is explained in the *Oracle Configurator Developer User's Guide*.

Publishing configuration models requires careful planning, based on a thorough understanding of the process by which publications of configuration models are defined and made available to host applications.

As part of your planning, consider the following:

- How will each publication be used?
- Which host application(s) need to access the publication?
- How will the configuration model be presented to the end user?
- How can the Oracle Configurator publication functionality help you achieve your deployment?
- Are you working with BOM Models or non-BOM Configurator Developer Models?

Once you have determined how the publication functionality applies to your situation, identify the necessary tasks in Oracle Applications and Oracle Configurator Developer .

Creating configuration models and publication requests is explained in the *Oracle Configurator Developer User's Guide* .

Designing A Project

Your project design should account for how you use host applications, Usages, effective date ranges, languages, publication modes, and database instances.

Consider the following:

- How many databases are you going to set up?

  For example, are you going to develop, test, and go live on only one database, or do you plan to develop test configuration models, but run your production environment on a separate, production database? For important things to consider regarding publishing and database instances, see Model Development, page 3-7.

- Are you going to use Usages to control a publication's availability?

  See Example: How a Usage Affects Model Structure, Rules, and Model Publications at Runtime, page 16-4.

- Are you going to use effective dates, Effectivity Sets, and Usages within configuration models to limit the availability of specific Model structure nodes or rules?

  For more information, see the chapter on effectivity in the *Oracle Configurator Developer User's Guide* .

- What host applications will access your publications?

- Is your host application registered in Oracle Applications?

  For information about registering applications, see the *Oracle E-Business Suite System Administrator's Guide*.

- Will you use the publication Mode to restrict access to testers and end users?

  For example, when testing on the production database before going live, setting the publication mode to Test excludes end users from accessing a publication, even though the publication still exists in the production database.

Preventing Publication Access Errors

To prevent end users from receiving errors, you should plan for and try to create publications for all circumstances in which host applications access your configuration models. Applications that can host a runtime Oracle Configurator can access different publications for a single configuration model. A publication corresponds to only one configuration model and one User Interface. A configuration model can have multiple User Interfaces and you can create many publications for the same Model.

How Host Applications Select a Published Model

All applications that can host a runtime Oracle Configurator select a specific Model publication to view by sending an initialization message to the Oracle Configurator Servlet. If a publication's applicability parameters match the parameters in this message, then the corresponding configuration model and UI appear in the Configurator window. If no matching publication is found but the Model was created from an imported BOM Model, then Oracle Configurator displays the BOM Model in the Generic Configurator UI. If no matching publication is found and the Model was created in Oracle Configurator, then Oracle Configurator displays an error.

For example, in your business you know that two different host applications, Oracle Order Management (OM) and Oracle *i*Store, will be used to configure Model M1. You define two unique UIs in Configurator Developer and create two publications for this Model. You set the Applications applicability parameter to Oracle Order Management for the first publication, and Oracle *i*Store for the second. An Oracle Applications user whose responsibility is assigned to Oracle Order Management selects Model M1 in the Sales Orders window, and clicks Configure.

Using the information in the initialization message, the OC Servlet selects the only publication in the database that:

- Has the Applications parameter set to Oracle Order Management

- Matches all of the other parameters specified in the initialization message

The OC Servlet then displays the configuration model and UI that you defined specifically for orders placed from Order Management in the Configurator window.

For detailed information about the initialization message, see Session Initialization, page 9-1.

For information about entering applicability parameters when creating a publication, see the *Oracle Configurator Developer User's Guide*.

Example: How a Usage Affects Model Structure, Rules, and Model Publications at Runtime

Your company makes and sells cars and has two types of Oracle Order Management users: experienced users, who are very familiar with each vehicle, and new users, who are either still in training or have worked for the company for only a short time.

The US Environmental Protection Agency (EPA) requires that cars sold in California meet more rigorous emissions standards than other states in the U.S. Therefore, cars that are sold in California must have different engine and exhaust components than cars sold elsewhere. Your experienced users need to be able to quickly configure orders and do not require much information except the state in which the customer lives. However, your less experienced users require more detailed information and guidance to consistently create valid, orderable configurations.

When defining the configuration model, you create additional Model structure, rules, and a UI to guide inexperienced users. The additional Model structure and rules provide the guided buying and selling questions to ensure that inexperienced users correctly configure each vehicle based on the state in which the customer lives. You then create a Usage called "Experienced User" and select this Usage for the guided buying or selling structure and rules in your Model.

Your System Administrator sets the profile option CZ: Publication Usage at the User level for each Oracle Configurator end user. For the experienced users, the System Administrator sets this profile option to "Experienced User". For inexperienced users, the System Administrator accepts the profile option's default value, which is "Any Usage."

You create two publications for the Model. One publication is intended for experienced users, so you select the appropriate UI and the Experienced User Usage when defining the publication's applicability parameters. The other publication is intended for inexperienced users, so you select the UI that has additional controls and information for configuring the car, but do not select a Usage (that is, you accept the default value, which is Any Usage).

When an end user wants to configure a car, Oracle Configurator checks how the CZ: Publication Usage profile option is set for that user, and applies this value to the configuration session. If the Usage specified is "Any Usage," then Oracle Configurator displays the publication and UI intended for the inexperienced user. This publication has additional UI controls, rules, and guided buying or selling questions to guide the user's selections.

If the Usage specified is "Experienced User," then Oracle Configurator displays the publication and UI intended for the experienced user. This publication has fewer rules and a very basic UI that enables the end user to select options and create a valid configuration very quickly.

Defining a Publication

This section explains:

Source and Remote Publications

Defining a publication in Oracle Configurator Developer creates a **source publication** with a unique publication ID in the CZ_MODEL_PUBLICATIONS table in the development database instance. When the publication and Model data is exported to the target database instance (by running a concurrent program), a record of the publication is created on the target database: this is called a **remote publication**. Each value in a source publication record corresponds to a value in the remote publication record. For details, see Data created when a configuration model is published, page 16-10. For details on creating a publication in Oracle Configurator Developer, see Configuration Model Publication Concurrent Programs, page C-15.

Do not confuse the term "remote publication" with the process of publishing a Model to a remote database instance. Creating a remote publication means creating a publication on an instance other than the one on which Configurator Developer is running. For more information, see Target Database Instance, page 16-8.

If a publication target instance is converted to a development instance, the source publication records are modified accordingly (see the Convert Publication Target Instance to Development Instance, page C-8 concurrent program for details). The publications on the source instance are marked as OBSOLETE, and the only action allowed on these publications is Delete.

When you define a publication record, Oracle Configurator Developer checks the source publication's attributes and applicability parameters to be sure they do not overlap with other source publications.

> **Warning:** Configurator Developer does not compare the source publication to any remote publications, even if the target database is the same database on which Configurator Developer is running. In other

words, the publishing process does not prevent users from publishing Models from multiple development instances to the same target instance. You can only be sure that you are not creating publications with overlapping applicability parameters in the same database (and possibly causing data synchronization errors) if you publish from a single development instance. For this reason, be sure that users publish configuration models from only *one* source database.

Tables Used in Publishing

The following database tables are used during the publishing process:

- CZ_EXT_APPLICATIONS
- CZ_MODEL_PUBLICATIONS
- CZ_MODEL_USAGES
- CZ_MODEL_USAGES_TL
- CZ_PB_CLIENT_APPS
- CZ_PB_LANGUAGES
- CZ_PB_MODEL_EXPORTS
- CZ_PUBLICATION_USAGES
- CZ_UI_ACTIONS
- CZ_UI_DEF

For detailed information about the publishing tables (or any other tables in the CZ schema), see the Oracle Integration Repository.

Publication Details

Access to a publication is determined in part by a publication's details and applicability parameters. When you create a new publication or edit an existing publication, these details are found in the Publications area of the Repository in Configurator Developer. A publication's details define the runtime circumstances and environment in which the published configuration model (that is, the publication) is available.

This section contains information about how the publication's details are used internally by the runtime Oracle Configurator. The publication details described are:

- Model , page 16-7

- Product ID, page 16-7
- User Interface, page 16-8
- Target Database Instance, page 16-8
- Mode, page 16-8

For general information about the publication attributes, including how to specify them when creating the publication record, see the *Oracle Configurator Developer User's Guide*.

Model

The Product ID column in the Publications area of the Workbench corresponds to the MODEL_KEY field in the CZ_MODEL_PUBLICATIONS table. This MODEL_KEY is the CZ_DEVL_PROJECTS.DEVL_PROJECT_ID that displays the CZ_DEVL_PROJECTS.NAME. This is the Model name that appears in the General areas of the Workbench in Configurator Developer.

Product ID

Product ID is a designation relevant when publishing in Oracle Configurator Developer. There is no corresponding Product node in a configuration model's structure.

The Product ID field in the Publications area of the Workbench displays different information depending on whether the specified Model is an imported BOM Model or a Oracle Configurator (non-BOM) Model.

If the configuration model is based on an imported BOM Model, the Product ID consists of the organization ID and Oracle Inventory Item ID, which are derived from Oracle Inventory (for example, 101 : 214738). This value is stored as the PRODUCT_KEY in CZ_MODEL_PUBLICATIONS, CZ_DEVL_PROJECTS, and CZ_IMP_DEVL_PROJECTS. In this case, the Product ID is read-only.

If the publication is based on a non-BOM Model that does not reference an imported BOM Model, and the PRODUCT_KEY field in CZ_DEVL_PROJECTS is not null, then that value is used in the publication record and is read-only. If the value is null, then the user enters a value.

If the publication is based on a non-BOM Model and *does* contain a Reference to a BOM Model, the Product ID consists of the imported BOM Model's Oracle Inventory Item ID and Organization ID. In this case, the Product ID is read-only.

> **Note:** If the Model you specified is a non-BOM Model, then the default Product ID is the name of the root Model node. For imported BOM Models, this value consists of the BOM Model's Item ID and Organization ID (defined in Oracle Inventory). You can change the Product ID when publishing a non-BOM Model; otherwise, it is

read-only.

The PRODUCT_KEY and the product_id parameter specified by the host application's session initialization message are the same. For more information about the session initialization message, see Session Initialization, page 9-1.

User Interface

If the configuration model specified by the publication has multiple User Interfaces, then the list of available User Interfaces on the Publications Repository page comes from the CZ_UI_DEFS table. The available User Interfaces are determined by the selected configuration model.

Target Database Instance

The list of values for this parameter includes all databases listed in the CZ_SERVERS table. This parameter indicates the database to which the publication and Model data are copied when you run one of the Configuration Model Publication Concurrent Programs, page C-15. The database you specify can be the same instance on which Configurator Developer is running, or a different one (that is, a remote database instance). However, Oracle strongly recommends that all source and target instances which participate in publishing be located on the same local area network. When publishing over a wide area network, performance can be degraded by network factors.

Before you can publish to a remote database instance, it must be defined and enabled. For details, see the Server Administration Concurrent Programs. , page C-9Note that the first time you create a publication on a remote database instance, the instance changes to a publication target, and Configurator Developer can no longer be accessed on that instance. To change it back to a development instance, run the Convert Publication Target Instance to Development Instance , page C-8concurrent program.

Mode

Values for this parameter include `Test, Production, or Disabled`. For information about the publication_mode parameter in the session initialization message, see Initialization Parameter Descriptions, page 9-17. See the *Oracle Configurator Installation Guide* for information on the Oracle Applications profile option CZ: Publication Lookup Mode.

Publication Applicability Parameters

Applicability parameters determine the availability of a publication to host applications. This section describes how the publication applicability parameters are used internally by the runtime Oracle Configurator. The applicability parameters are:

- Applications, page 16-9

- Languages, page 16-9
- Usages, page 16-9
- Date Range, page 16-10

For general information about applicability parameters, including how to specify them when publishing, see the *Oracle Configurator Developer User's Guide*. For more information about how a host application interacts with these parameters to select a publication, see Model Publication Identification Parameters, page 9-13.

Applications

When creating a publication, the entries in the CZ_EXT_APPLICATIONS table appear in Applications list of values on the Publications page. These entries are host applications that support Oracle Configurator as well as any application that an Oracle Configurator Administrator has added to the CZ_EXT_APPLICATIONS table.

If an application does not appear in the Applications list, and you want to make publications available to that application, then the Oracle Configurator Administrator can add it to the list by running the Add Application to Publication Applicability List, page C-9 concurrent program. For more information about the CZ_EXT_APPLICATIONS table, see the Oracle Integration Repository.

When you save a publication, the specified applications and publication ID are stored in the CZ_PB_CLIENT_APPS table.

Languages

The Languages applicability parameter is stored in the LANGUAGE column in CZ_MODEL_APPLICABILITIES_V. The Language list of values is retrieved from the FND_LANGUAGES table.

For information about Multiple Language Support (MLS), see Multiple Language Support, page 14-1.

Usages

The Usages defined in Oracle Configurator Developer are stored in the following tables:

- CZ_MODEL_USAGES contains the numeric Usage ID assigned by the database (MODEL_USAGE_ID) and the Usage Name that you assign when you create the Usage (NAME).
- CZ_MODEL_USAGES_TL contains the translatable Usage Descriptions that you can create in each installed language (DESCRIPTION).

The Usage Names are displayed in the list of values when assigning Usages to a publication on the Model Publication page. The Usages assigned to a publication are

stored in CZ_PUBLICATION_USAGES.

For an example of how Usages are used by a host application at runtime, see Example: How a Usage Affects Model Structure, Rules, and Model Publications at Runtime, page 16-4.

For general information about Usages and how to define them in Configurator Developer, see the *Oracle Configurator Developer User's Guide*.

Date Range

A publication's effective dates are stored in the columns APPLICABLE_FROM and APPLICABLE_UNTIL in the CZ_MODEL_PUBLICATIONS table.

Publishing a Configuration Model

After defining a source publication in Oracle Configurator Developer, the configuration model data must be copied to the target database by doing one of the following:

- Submitting a concurrent program request through Oracle Applications. For more information, see Configuration Model Publication Concurrent Programs, page C-15 .

  When you submit an Oracle Applications concurrent request to publish Model data to a target database, the Model, any referenced Models, and any referenced UI Content Templates must either be unlocked or locked by you. For more information on locking, see the *Oracle Configurator Developer User's Guide*.

- Using the `cz_modeloperations_pub.publish_model` API through SQL*PLUS. For more information, see Programmatic Tools for Maintenance, page 18-1.

- Running a batch process.

Each of these tasks create a remote publication on the target database. When the publication completes successfully, the remote publication can be accessed by host applications such as Oracle Order Management or *i*Store. The CZ_MODEL_PUBLICATIONS table stores the high level information about the publication. A new entry is entered into the CZ_DEVL_PROJECT table. For table details see the Oracle Integration Repository.

Data created when a configuration model is published, page 16-10 shows some of the data that is created when a configuration model is published.

Data created when a configuration model is published

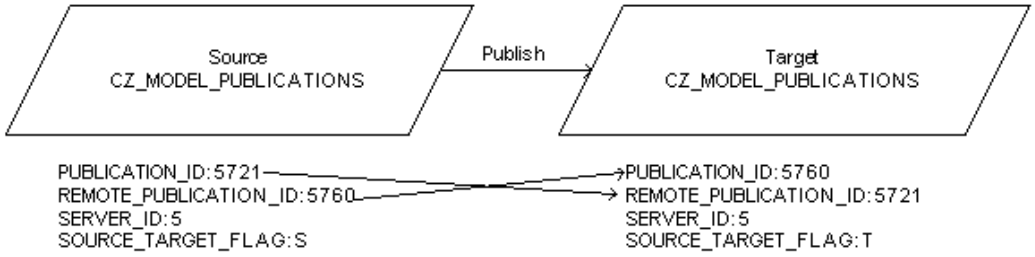
- Source publication record:
  - PUBLICATION_ID: 5721
  - SERVER_ID: 5

    - REMOTE_PUBLICATION_ID:5760
    - SOURCE_TARGET_FLAG: S
- Corresponding remote publication record:
    - PUBLICATION_ID: 5760
    - SERVER_ID: 5
    - REMOTE_PUBLICATION_ID:5721
    - SOURCE_TARGET_FLAG: T

Illustration of a Publication Record Mapping, page 16-11 illustrates how the source and target publication records have corresponding values in the database. This correspondence allows source and target publications to be matched when updating or synchronizing the publication data.

Illustration of a Publication Record Mapping, page 16-11 illustrates how PUBLICATION_ID 5721 in the source publication corresponds to REMOTE_PUBLICATION_ID 5721 in the target publication. The illustration also shows that REMOTE_PUBLICATION_ID 5760 in the source publication corresponds to PUBLICATION_ID 5760 in the target publication. The source publication's SERVER_ID is 5, which corresponds to the SERVER_ID entry in the CZ_SERVERS table (whose value is also 5).

Illustration of a Publication Record Mapping



In the source database instance, the SERVER_ID column in the CZ_SERVERS table identifies the target's SERVER_ID. This same column and table on the target database instance is the target's SERVER_ID (not the source's SERVER_ID).

For more information about defining publications, examples of overlapping publications, and UI Templates, see the *Oracle Configurator Developer User's Guide*.

For more information, see Configuration Model Publication Concurrent Programs, page C-15.

Publication Profile Options

If a Usage or publication mode is not specified in the session initialization message, then the following profile options provide default values for these parameters:

- CZ: Publication Usage
- CZ: Publication Lookup Mode

Publishing and Model References

If you are publishing a configuration model that has References to other Models, then all of the referenced Models are also copied to the target database and are part of the publication. If a referenced Model itself is not published, then it can only be configured as part of its parent (the published Model). In other words, an end user can configure only Models that have been published.

The availability of referenced Models is controlled by the Usages and Date Range applicability parameters. See the *Oracle Configurator Developer User's Guide* for more information on the Usages and Date Range applicability parameters.

Copying User Interface Data

The runtime Oracle Configurator UI supports the use of UI Templates and generated User Interfaces. Publishing a configuration model copies the following UI-specific data:

- Database records in the following tables that have UI_DEF_ID as part of the primary key in the target database instance:
  - CZ_UI_ACTIONS
  - CZ_UI_CONT_TYPE_TEMPLS
  - CZ_UI_DEFS
  - CZ_UI_PAGES
  - CZ_UI_PAGE_REFS
  - CZ_UI_PAGE_SETS
  - CZ_UI_REFS
  - CZ_UI_TEMPLATES
- Generated User Interfaces for a given UI_DEF_ID and listed in the following:

- CZ_UI_CONT_TYPE_TEMPLS
- CZ_UI_PAGES.jrad_doc
- CZ_UI_TEMPLATES.jrad_id

All translations are stored in the JRAD repository and are copied to the target database when the generated UI is copied.

Copying Model Rules

By default, the publishing process copies all configuration model data to the target database. You can control whether rules defined in Configurator Developer are copied using the PublishingCopyRules, page 4-20 setting in the CZ_DB_SETTINGS table. This setting does not affect Configurator Extension Rules; all Configurator Extension Rules are always copied when you publish or republish a configuration model.

For more information about the PublishingCopyRules setting, see PublishingCopyRules, page 4-20.

Checking BOM Model and Configuration Model Similarity

When you are publishing to a remote server, the publication concurrent programs call the Model synchronization concurrent programs. If there are key discrepancies between the source BOM Model and the configuration model to be published, such as the Items on both Models are not the same, then an error message is logged by the publication concurrent program and the configuration model is not published.

Publishing Error when Checking BOM Model and Configuration Model, page 16-13 illustrates an error found in CZ_DB_LOGS file when attempting to publish a configuration model (publication ID = 28261).

Publishing Error when Checking BOM Model and Configuration Model

```
Unable to proceed with publishing because the configuration model
'SOURCE MODEL1-Pub Synch(204 501069)' does not match the corresponding
bill on the target server. The model has not been published.
     28261     36638
BOM Synchronization, version 115.29, started 2002/12/18/16:27:41,
session run ID: 36639
     28261     36638
Maximum quantity does not match for item 'ATO OC6' with parent 'ATO
Model4' in configuration model '
SOURCE MODEL1=>PTO Model2=>ATO Model3=>ATO Model4'
     28261     36638
Process terminated for publication_id: 28261
     28261     36638
```

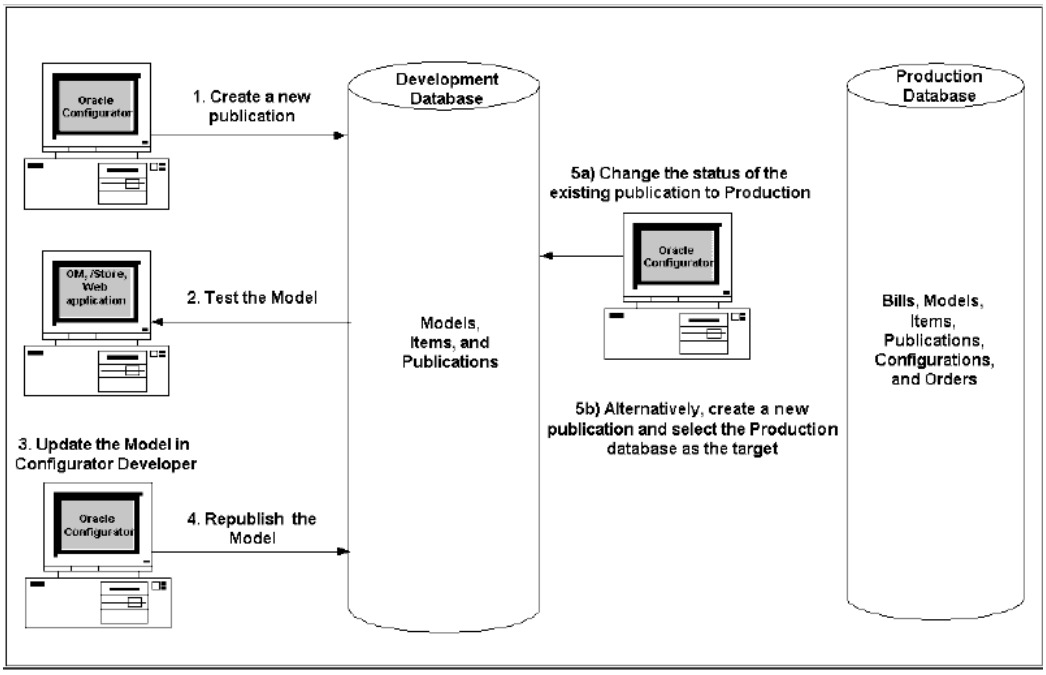
For more synchronization information, see The BOM Model Synchronization Process, page 7-2.

Maintaining Publications

Typically, a configuration model may undergo many iterations of testing and updates before it is made available to customers in a production environment. Publishing gives you complete control over each step in a configuration model's lifecycle, enabling you to maintain and update Models that are under development while making approved versions available in your production environment.

The following illustration shows an overview of the publication process, in which a user creates a new publication from Configurator Developer on the Development database, tests the Publication, updates the Model in Configurator Developer, and republishes the Model. Once the publication has been tested thoroughly, a Developer user changes the Mode applicability parameter to Production, or creates a new publication and selects the Production database as its target.

Example of the Publication Process



Publication Status

The operations you can perform on an existing publication depend on its current status. You can view detailed information about publications, including their status, on the Model Publication page in Configurator Developer.

The table Publication Status and Valid Operations, page 16-15 lists each status and the corresponding tasks you can perform.

Publication Status and Valid Operations

Status	New or New Copy	Edit	Republish	Delete	Disabled	Edit UI
Complete	Y	Y	Y	Y	Y	N
Pending	Y	Y	N	Y	Y	N
Update Pending	Y	N	N	N	N	N
Processing	Y	N	N	N	N	N
Error	Y	N	N	Y	N	N
Obsolete	Y	N	N	Y	N	N

Configurator Developer updates the status of all publications whenever you navigate to the Publication Repository page or click the Browser Refresh. The **Status** column may change, for example, when one of the publication concurrent programs completes successfully.

Following is a description of each publication status:

- **Complete**: The Oracle Applications concurrent program successfully copied the configuration model to the publication target database.

- **Pending**: A request to create a new publication has been created in Configurator Developer. When the Oracle Applications concurrent program successfully copies the Model data to the publication target database, the pending status changes to Complete. If an error occurs during the publication concurrent program, then the publication's status changes to Error.

- **Pending Update**: A request to update the existing publication has been created. When the Oracle Applications concurrent program successfully copies the Model data to the publication target database, the Pending Update status changes to Complete. If an error occurs during the update, then the publication's status rolls back to Complete so that the user can republish the Model.

- **Processing**: The Oracle Applications concurrent manager is processing a request to create or update this publication.

- **Error**: An error occurred while processing the request to create or update this publication. An error can occur, for example, when you create a new source

publication but another Configurator Developer user updates the Model before the Oracle Applications concurrent program is complete.

- **Obsolete:** When a publication's target instance is converted to a source development instance, all publications on that target database instance for publishing are marked as Obsolete in the original source instance. A copy of the obsolete publication can be made, but the publication details page will not list the original publication's target server. You must choose a new target server.

Editing Publications

When an Oracle Configurator Developer user edits a publication, the changes are automatically propagated to the remote publication in the CZ_MODEL_PUBLICATIONS table (in the target database).

Depending on the changes made in Oracle Configurator Developer, editing the publication may involve adding or deleting records in the CZ_PB_CLIENT_APPS or CZ_PUBLICATION_USAGES tables, or changing the publication's mode or valid date range.

For information on how to edit a publication, see the *Oracle Configurator Developer User's Guide*.

> **Note:** If you publish a new version of the Model and there are previous published versions in memory because you are still running on the same Apache JServ, users could get out of memory errors if the max heap size can't accommodate all of the published Models in memory. You can increase the maximum heap size (which could degrade performance) or bounce Apache to clear the previous publication out of memory. Oracle Applications Java Caching Framework provides the ability to manage the caching and decaching of Model and UI data, which optimizes both runtime performance and memory management. For more information, see the *Oracle Configurator Performance Guide*.

Disabling, Deleting, and Re-enabling Publications

You can make a publication unavailable to host applications by disabling it in the Publication Repository. When a publication is disabled, it remains listed in the Publication Repository, its status does not change, but the publication's Disabled column notes that the publication has been disabled. When a publication is disabled you can modify its applicability parameters or re-enable it.

You can also delete a publication. When you delete a publication, it no longer appears in the Publication Repository page in Oracle Configurator Developer, and it cannot be recovered. However, the publication record still exists in the CZ schema until the Purge Configurator Tables concurrent program is run. For more information on the Purge

Configurator Tables concurrent program, see Purge Configurator Tables, page C-4.

See the *Oracle Configurator Developer User's Guide* for more information on disabling, deleting and re-enabling publications.

Republishing

This section describes the database tables that are updated when you republish a configuration model. For information about how to republish a configuration model in Configurator Developer, see the *Oracle Configurator Developer User's Guide*.

When an Oracle Configurator Developer user republishes a configuration model, the following occurs:

1. The status of the original publication changes to PUP (Pending Update) in the Publication Repository, and STATUS is PUP in the CZ_PB_MODEL_EXPORTS table. The publication status does not change until one of the publication concurrent programs completes successfully.

2. A new publication record is created in the CZ_MODEL_PUBLICATIONS, CZ_PB_CLIENT_APPS, and CZ_PUBLICATION_USAGES tables of the publication source development instance. This publication record has the same applicability parameters as the original publication.

   Note: If you set the profile option CZ: Populate Decimal Quantity Flags to `Yes` and then reimport or refresh your BOM Models, you must republish existing Model publications to ensure that they use the new setting. Decimal quantities are explained in Importing Decimal or Integer Quantities, page 5-13.

   Note: If a new language has been added to Oracle Applications, then you must republish your Models in order for the User Interface labels to be displayed in the new foreign language. For more information on MLS, see Multiple Language Support, page 14-1.

Determining Publishing Information

Knowing the UI_DEF_ID can be helpful when you want to look up information about a publication using SQL*Plus. Using the Publication ID from Oracle Configurator Developer's Publication Repository in a simple SQL*Plus query returns the UI_DEF_ID. The UI_DEF_ID can then be used in queries on the CZ_CONFIG_HDRS, CZ_MODEL_PUBLICATIONS, CZ_UI_DEFS, CZ_UI_NODES, CZ_UI_NODE_PROPS, CZ_UI_PROPERTIES.

Query for UI_DEF_ID

```
select ui_def_id
from cz_model_publications
where publication_id=publication number ;
  UI_DEF_ID
     2760
```

UI_DEF_ID can also be found in CZ_UI_DEFS, or by calling the PL/SQL function `cz_cf_api.ui_for_item`. For more information about this function, see Reference for the CZ_CF_API and the CZ_CONFIG_API_PUB Packages, page 17-11.

Retrieving Orders from Previously Published Models

A situation may develop where you want to retrieve prior orders that were placed against a previously published Model, rather than the more recent Model that has new structure and new rules. For example, when the first Model was published the **From** and **To** Date Range applicability parameters were not specified.

To retrieve orders for the previously published Model, you must:

1. Edit the first published Model's **Date Range** applicability parameter to have an end date.
2. Republish the Model.
3. Publish the newer Model with the **From** Date Range applicability parameter equal to the **To** Date Range of the first published Model.

> **Note:** If a previously published configuration model is modified in Configurator Developer and is then republished, then end users can restore any saved configurations that were created using the original publication. However, if the Model's structure or rules have changed, the end user may need to make additional selections to create a valid and complete configuration.

See the *Oracle Configurator Developer User's Guide* to learn how to perform these tasks.

Synchronizing Publication Data

Publication data must be synchronized whenever you:

- Clone a publication source or target database instance
- Migrate data from one database instance to another

For more information, see Synchronizing Data, page 7-1.

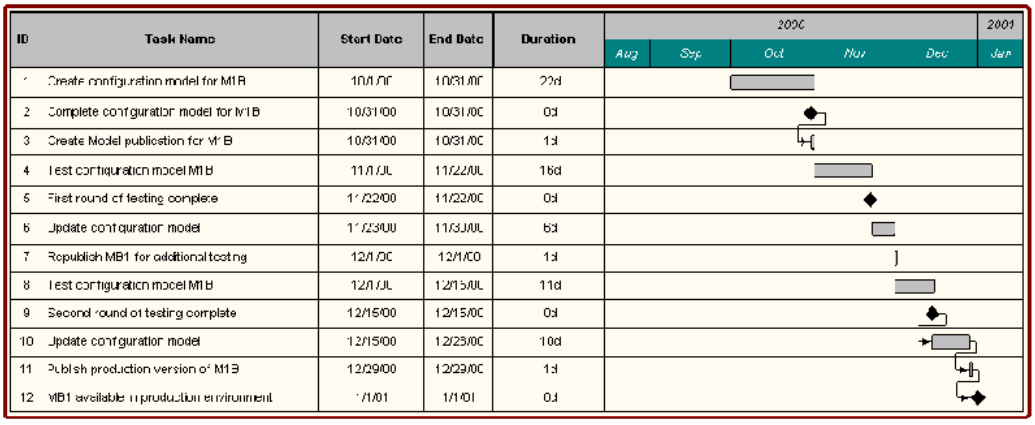
Example of Maintaining Publications

This section provides an example of how a business may develop configuration models and maintain publications in a development environment. An organization has a laptop computer called M1A that is currently in production. However, a new version of M1A is under development and this computer, M1B, will replace M1A by the end of the year. The new Model must replace the older version in the production environment and there can be no period of time when neither is available to customers.

Maintaining Publications, page 16-19 provides an overview of how this organization plans to develop, test, and release M1B into production. It is a time line that lists the typical activities involved in maintaining a Model publication. Details of each step and a description of the table are provided in the text following the table.

Maintaining Publications

ID	Task Name	Start Date	End Date	Duration
1	Create configuration model for M1B	10/1/00	10/31/00	22d
2	Complete configuration model for M1B	10/31/00	10/31/00	0d
3	Create Model publication for M1B	10/31/00	10/31/00	1d
4	Test configuration model M1B	11/1/00	11/22/00	16d
5	First round of testing complete	11/22/00	11/22/00	0d
6	Update configuration model	11/23/00	11/30/00	6d
7	Republish MB1 for additional testing	12/1/00	12/1/00	1d
8	Test configuration model M1B	12/1/00	12/15/00	11d
9	Second round of testing complete	12/15/00	12/15/00	0d
10	Update configuration model	12/15/00	12/25/00	10d
11	Publish production version of M1B	12/29/00	12/29/00	1d
12	MB1 available in production environment	1/1/01	1/1/01	0d



Details

The following steps correspond to the ID column in the project schedule shown in Maintaining Publications, page 16-19.

1. Using Configurator Developer, the development team creates a new configuration model (M1B) to reflect the new product's features and enhancements. The Model is unit tested periodically in Oracle Configurator Developer, but it is not yet made available for system testing.

2. The new configuration model is complete and ready for system testing.

3. Developers create publication P1 and sets its publication Mode to Test. The publication is effective immediately and no end date is required because it can be modified at any time. The Applications and Usages parameters specify which host applications and end users can access the Model.

4. The quality assurance (QA) group accesses and tests the configuration model for

product M1B and reports any problems to the development group. The host application that the testers use selects the configuration model to display based on the applicability parameters defined for publication P1.

5. The first round of testing configuration model M1B is complete.

6. Developers incorporate comments from testers by updating the configuration model in Configurator Developer. This may include building new Model structure, creating or modifying rules, or updating the User Interface.

7. When changes to the Model are complete, developers republish the Model. Republishing copies any new or modified data to the specified database so that the QA group can begin a second round of testing. Republishing does not change any of the original applicability parameters, so publication P1 is available to the same host applications and users as in the first round of testing.

8. The QA group performs a second round of testing Model M1B.

9. The second round of testing is complete and additional comments are reported to the development group.

10. Developers update the configuration model in Configurator Developer.

11. Company management and the development group agree that the configuration model is ready for production. In this enterprise, the development and production environments exist on the same database, so a developer makes the product available to customers by modifying the applicability parameters of the existing publications as follows:

    1. Change the publication Mode P1 from Test to Production

    2. Change the **To** Effectivity Date of the now obsolete publication for Model M1A to 12:00:00 a.m. on 01/01/01

    3. Specify a **From** Effectivity Date for publication P1 of 12:00:00 a.m. on 01/01/01

    This modification ensures that there is no gap in the availability of the old and new products because M1A becomes obsolete at the same time M1B becomes available in production.

See the *Oracle Configurator Developer User's Guide* for more information.

17

Programmatic Tools for Development

This chapter describes a set of programmatic tools (PL/SQL procedures and functions) that may be useful in developing a configuration model and deploying a runtime Oracle Configurator.

This chapter covers the following topics:

- Overview
- Overview of the CZ_CF_API and CZ_CONFIG_API_PUB Packages
- Choosing the Right Tool for the Job
- Reference for the CZ_CF_API and the CZ_CONFIG_API_PUB Packages
- COMMON_BILL_FOR_ITEM
- CONFIG_MODEL_FOR_ITEM
- CONFIG_MODELS_FOR_ITEMS
- CONFIG_MODEL_FOR_PRODUCT
- CONFIG_MODELS_FOR_PRODUCTS
- CONFIG_UI_FOR_ITEM
- CONFIG_UI_FOR_ITEM_LF
- CONFIG_UI_FOR_PRODUCT
- CONFIG_UIS_FOR_ITEMS
- CONFIG_UIS_FOR_PRODUCTS
- COPY_CONFIGURATION
- CZ_CONFIG_API_PUB.COPY_CONFIGURATION
- COPY_CONFIGURATION_AUTO
- CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO
- DEFAULT_NEW_CFG_DATES

- DEFAULT_RESTORED_CFG_DATES
- DELETE_CONFIGURATION
- ICX_SESSION_TICKET
- MODEL_FOR_ITEM
- MODEL_FOR_PUBLICATION_ID
- POOL_TOKEN_FOR_PRODUCT_KEY
- PUBLICATION_FOR_ITEM
- PUBLICATION_FOR_PRODUCT
- PUBLICATION_FOR_SAVED_CONFIG
- REGISTER_MODEL_TO_POOL
- UNREGISTER_MODEL_FROM_POOL
- UNREGISTER_POOL
- UI_FOR_ITEM
- UI_FOR_PUBLICATION_ID
- VALIDATE
- CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION

Overview

This chapter describes programmatic tools that you can use primarily to develop a configuration model and deploy a runtime Oracle Configurator. This includes:

- Choosing the Right Tool for the Job, page 17-7
- Reference for the CZ_CF_API and the CZ_CONFIG_API_PUB Packages, page 17-11

Important: For the latest reference information on these APIs, see the Oracle Integration Repository, which is installed with your patched instance of the Oracle E-Business Suite, as described in the Preface of this guide. In the Integration Repository, the package described in this chapter can be located by using the Search function on the Internal Name `CZ_CF_API`.

Important: This chapter includes references to DHTML user interfaces, but these are temporarily retained for historical informational purposes only. As of this release, DHTML UIs are no

longer supported.

For information on tools for maintaining a deployed runtime Oracle Configurator, see Programmatic Tools for Maintenance, page 18-1.

Overview of the CZ_CF_API and CZ_CONFIG_API_PUB Packages

The programmatic tools that you use while developing or deploying a runtime Oracle Configurator are provided in the PL/SQL packages CZ_CF_API and CZ_CONFIG_API_PUB.

Purpose of the Packages

The CZ_CF_API package contains a set of APIs that enable you to perform various tasks such as the following:

- Copying and deleting configurations that are not networked configurations
- Determining default dates used by the runtime Oracle Configurator
- Establishing session identity
- Identifying publications
- Validating configurations
- Verifying configurations

The CZ_CONFIG_API_PUB package contains a set of APIs that enable you to copy configurations including networked configurations and view an existing configuration in the CZ schema.

Overview of Procedures and Functions

The table Overview of Procedures and Functions in the Package CZ_CF_API, page 17-4 summarizes and categorizes the procedures and functions available in the packages CZ_CF_API and CZ_CONFIG_API_PUB. The column labeled P/F indicates whether an API is a procedure or a function.

These procedures and functions are described in individual detail in Reference for the CZ_CF_API and the CZ_CONFIG_API_PUB Packages, page 17-11.

Overview of Procedures and Functions in the Package CZ_CF_API

Category	API Name	P/F
Working with Common Bills, page 17-8	COMMON_BILL_FOR_ITEM, page 17-14	P
Copying and Deleting Configurations, page 17-7	COPY_CONFIGURATION, page 17-37	P
	CZ_CONFIG_API_PUB.COPY_CONFIGURATION, page 17-39	
	COPY_CONFIGURATION_AUTO, page 17-42 CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO, page 17-45	P
	DELETE_CONFIGURATION, page 17-51	P
Setting Configuration Dates, page 17-7	DEFAULT_NEW_CFG_DATES, page 17-48	P
	DEFAULT_RESTORED_CFG_DATES, page 17-49	P
Establishing Session Identity, page 17-7	ICX_SESSION_TICKET, page 17-53	F
Identifying Publications, page 17-8	CONFIG_MODEL_FOR_ITEM, page 17-15	F
	CONFIG_MODEL_FOR_PRODUCT, page 17-19	F
	CONFIG_MODELS_FOR_ITEMS, page 17-17	F
	CONFIG_MODELS_FOR_PRODUCTS, page 17-21	F
	CONFIG_UI_FOR_ITEM, page 17-23	F
	CONFIG_UI_FOR_ITEM_LF, page 17-26	F
	CONFIG_UI_FOR_PRODUCT, page 17-29	F
	CONFIG_UIS_FOR_ITEMS, page 17-31	F

Category	API Name	P/F
	CONFIG_UIS_FOR_PRODUCTS, page 17-34	F
	MODEL_FOR_ITEM, page 17-54	F
	MODEL_FOR_PUBLICATION_ID, page 17-56	F
	PUBLICATION_FOR_ITEM, page 17-58	F
	PUBLICATION_FOR_PRODUCT, page 17-59	F
	PUBLICATION_FOR_SAVED_CONFIG, page 17-61	F
	UI_FOR_ITEM, page 17-66	F
	UI_FOR_PUBLICATION_ID, page 17-68	F
Routing Models to Specified JVMs, page 17-11	REGISTER_MODEL_TO_POOL, page 17-63	P
	UNREGISTER_MODEL_FROM_POOL, page 17-64	P
	UNREGISTER_POOL, page 17-65	P
	POOL_TOKEN_FOR_PRODUCT_KEY, page 17-57	F
Validating Configurations, page 17-7	VALIDATE, page 17-69	P
Verifying Configurations, page 17-7	CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION, page 17-72	P

Installation of the Packages

These packages are installed in the Oracle Applications database as part of Oracle Configurator.

- If you installed a new instance of Oracle Applications, then these packages were installed by using Oracle Rapid Install.
- If you installed Oracle Configurator in an existing instance of Oracle Applications, then these packages were installed by applying the appropriate Oracle Configurator

patch.

See the *Oracle Configurator Installation Guide* for details about installing Oracle Configurator.

References for Working with PL/SQL Procedures and Functions

For background information and details on basic aspects of working with the PL/SQL procedures and functions in this package, refer to the table below, References for Working with PL/SQL Procedures and Functions, page 17-6.This table lists relevant topics in the Oracle documentation library.

References for Working with PL/SQL Procedures and Functions

For this topic ...	See this reference document area ...
• User-defined data types • Procedures and packages	Oracle database concepts
• Using procedures and packages • Calling stored procedures • Understanding the Oracle programmatic environments	Oracle Application's developer's guide fundamentals
• Language elements • Packages • Index-by tables • Collections and records • User-defined subtypes	PL/SQL user's guide and reference
• Using SQL*Plus	SQL*Plus user's guide and reference
• UTL_HTTP	Oracle supplied PL/SQL packages reference

Choosing the Right Tool for the Job

These procedures and functions are described in detail in Procedures and Functions in the CZ_CF_API and CZ_CONFIG_API_PUB Packages, page 17-12.

Establishing Session Identity

Use the following function to establish the identity of a Oracle Applications database session:

- ICX_SESSION_TICKET, page 17-53

Setting Configuration Dates

Use these procedures to determine the dates used for configurations:

- DEFAULT_NEW_CFG_DATES, page 17-48
- DEFAULT_RESTORED_CFG_DATES, page 17-49

Validating Configurations

Use this procedure to validate a configuration:

- VALIDATE, page 17-69

Verifying Configurations

Use this procedure to verify that the configuration exists and that it is both valid and complete:

- CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION, page 17-72

Copying and Deleting Configurations

Use these procedures to copy and delete configurations:

- COPY_CONFIGURATION, page 17-37 - not to be used with networked configurations
- COPY_CONFIGURATION_AUTO, page 17-42 - not to be used with networked configurations
- CZ_CONFIG_API_PUB.COPY_CONFIGURATION, page 17-39 - used with networked configurations

- CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO, page 17-45 - used with networked configurations
- DELETE_CONFIGURATION, page 17-51

Working with Common Bills

Use this procedure to retrieve a common bill:

- COMMON_BILL_FOR_ITEM, page 17-14

Identifying Publications

After publishing Models, you can verify whether a publication lookup will succeed for a given set of applicability parameters. See Applicability Parameters, page 17-9 for details about specifying applicability parameters.

Functions for Identifying Publications

Use these functions to look up publications for a given set of applicability parameters:

- CONFIG_MODEL_FOR_ITEM, page 17-15
- CONFIG_MODEL_FOR_PRODUCT, page 17-19
- CONFIG_MODELS_FOR_ITEMS, page 17-17
- CONFIG_MODELS_FOR_PRODUCTS, page 17-21
- CONFIG_UI_FOR_ITEM, page 17-23
- CONFIG_UI_FOR_ITEM_LF, page 17-26
- CONFIG_UI_FOR_PRODUCT, page 17-29
- CONFIG_UIS_FOR_ITEMS, page 17-31
- CONFIG_UIS_FOR_PRODUCTS, page 17-34
- MODEL_FOR_ITEM, page 17-54
- MODEL_FOR_PUBLICATION_ID, page 17-56
- PUBLICATION_FOR_ITEM, page 17-58
- PUBLICATION_FOR_PRODUCT, page 17-59
- PUBLICATION_FOR_SAVED_CONFIG, page 17-61

- UI_FOR_ITEM, page 17-66
- UI_FOR_PUBLICATION_ID, page 17-68

Applicability Parameters

Applicability parameters control the availability of a publication in your development or production environment

You can use applicability parameters in Oracle Configurator Developer (OCD) to determine which Model and UI to display when you publish a Model. See the *Oracle Configurator Developer User's Guide* for more information about applicability parameters and publishing.

You can also use applicability parameters in the initialization message that a host application sends to the Oracle Configurator Servlet. See Session Initialization, page 9-1 for more information.

The table Applicability Parameters for Publication Searches, page 17-9 lists the applicability parameters in the CZ_CF_API package that many of the functions and procedures in this package use to search for Models, UIs, and publications.

This table lists each parameter's data type, the corresponding field in the Model Publishing window in Oracle Configurator Developer, and a describes each parameter.

Applicability Parameters for Publication Searches

Parameter in this package	Data type	Parameter as it appears in Configurator Developer	Description
calling_application_id	number	Applications	The registered ID of an application for which the Model is published. This is a valid APPLICATION_ID from FND_APPLICATION. Example value: `660`
config_lookup_date	date	Date (From, To)	Provide a date that falls inside the applicable range for the publication. Use the standard Oracle TO_DATE function to format the date.

Parameter in this package	Data type	Parameter as it appears in Configurator Developer	Description
language	varchar2	Languages	Language code for an installed language (such as 'US'). CZ_PB_LANGUAGES is accessed to identify the publication assigned to the specified language. The default is NULL. If the parameter is NULL, then userenv("LANG") determines the language. Example value: `'US'`
product_key	varchar2	Product ID	For imported models, the product_key is the ORGANIZATION_ID concatenated with the INVENTORY_ITEM_ID, in MTL_SYSTEMS_ITEMS. For Models created in Oracle Configurator Developer, the Product ID is generated from the name of the Model when you publish the Model. Example value (for an imported Model): `204:2510`
publication_mode	varchar2	Mode	The publication mode for the publication. Values are 'P' (production) or 'T' (test). The default is NULL. If NULL, then the CZ: Publication Lookup Mode profile option value is checked. Example value: `'T'`
usage_name	varchar2	Usages	Name of a Usage defined in Oracle Configurator Developer. If this is NULL, then the CZ: Publication Usage profile option value is checked. Example value: `'my usage'`

List Parameters

In order to reduce the number of function calls when an application must find Models for multiple products or items, some functions in this package take parameters that are *lists* of values, and return a list of values (as identified in the syntax for the function). To

pass a list of values, this package defines several custom data types that are collections.

Parameters in this package that are of one of these list types do not default to NULL.

See Custom Data Types, page 17-11 for the definition of these types.

Routing Models to Specified JVMs

Use these procedures and functions to register and unregister Models in the pool mapping table. See Routing Models to Specified JVMs, page 20-10 for background.

- REGISTER_MODEL_TO_POOL, page 17-63
- UNREGISTER_MODEL_FROM_POOL, page 17-64
- UNREGISTER_POOL, page 17-65
- POOL_TOKEN_FOR_PRODUCT_KEY, page 17-57

Reference for the CZ_CF_API and the CZ_CONFIG_API_PUB Packages

- This section provides descriptions of each of the procedures and functions in the CZ_CF_API and CZ_CONFIG_API_PUB packages. These procedures and functions are listed alphabetically in Procedures and Functions in the Packages CZ_CF_API and CZ_CONFIG_API_PUB, page 17-13
- Descriptions of the custom data types defined in the package are provided in Custom Data Types, page 17-11.
- For a basic example of how to call one of the functions in the CZ_CF_API package, see Using the UI_FOR_PUBLICATION_ID Function, page 17-69.
- See Overview of the CZ_CF_API and CZ_CONFIG_API_PUB Packages, page 17-3.

Custom Data Types

Custom Data Types in the Package CZ_CF_API, page 17-12 describes the custom data types that are defined in this package.

- For background on the record data type, see the references for collections and records.
- For background on the table data type, see the references for collections.
- For background on subtypes, see the references for user-defined subtypes.
- For background on the UTL_HTTP package, see the references for UTL_HTTP.

For background on these custom data types, see the references under References for Working with PL/SQL Procedures and Functions, page 17-6:

The following table includes the custom data types and provides a description of each.

Custom Data Types in the Package CZ_CF_API

Custom Type	Description
INPUT_SELECTION	Record consisting of: COMPONENT_CODE VARCHAR2(1200) QUANTITY NUMBER INPUT_SEQ NUMBER CONFIG_ITEM_ID DEFAULT NULL
CFG_INPUT_LIST	Table of INPUT_SELECTION, page 17-12 indexed by BINARY_INTEGER
CFG_OUTPUT_PIECES	This is a result of the batch validation message. Subtype of UTL_HTTP.HTML_PIECES. It is a table of VARCHAR2(2000).
NUMBER_TBL_TYPE	Table of NUMBER
DATE_TBL_TYPE	Table of DATE
VARCHAR2_TBL_TYPE	Table of VARCHAR2(255)

Procedures and Functions in the CZ_CF_API and CZ_CONFIG_API_PUB Packages

This section provides descriptions of each of the procedures and functions in the CZ_CF_API and CZ_CONFIG_API_PUB packages, arranged alphabetically. These procedures and functions are listed alphabetically in Procedures and Functions in the Packages CZ_CF_API and CZ_CONFIG_API_PUB, page 17-13.

The following table lists the API procedures and functions in the CZ_CF_API package. The column labeled P/F indicates whether an API is a procedure or a function.

Procedures and Functions in the Packages CZ_CF_API and CZ_CONFIG_API_PUB

API Name	P/F
COMMON_BILL_FOR_ITEM, page 17-14	P
CONFIG_MODEL_FOR_ITEM, page 17-15	F
CONFIG_MODEL_FOR_PRODUCT, page 17-19	F
CONFIG_MODELS_FOR_ITEMS, page 17-17	F
CONFIG_MODELS_FOR_PRODUCTS, page 17-21	F
CONFIG_UI_FOR_ITEM, page 17-23	F
CONFIG_UI_FOR_ITEM_LF, page 17-26	F
CONFIG_UI_FOR_PRODUCT, page 17-29	F
CONFIG_UIS_FOR_ITEMS, page 17-31	F
CONFIG_UIS_FOR_PRODUCTS, page 17-34	F
COPY_CONFIGURATION, page 17-37	P
COPY_CONFIGURATION_AUTO, page 17-42	P
CZ_CONFIG_API_PUB.COPY_CONFIGURATION, page 17-39	P
CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO, page 17-45	P
DEFAULT_NEW_CFG_DATES, page 17-48	P
DEFAULT_RESTORED_CFG_DATES, page 17-49	P
DELETE_CONFIGURATION, page 17-51	P
ICX_SESSION_TICKET, page 17-53	F
MODEL_FOR_ITEM, page 17-54	F

API Name	P/F
MODEL_FOR_PUBLICATION_ID, page 17-56	F
POOL_TOKEN_FOR_PRODUCT_KEY, page 17-57	F
PUBLICATION_FOR_ITEM, page 17-58	F
PUBLICATION_FOR_PRODUCT, page 17-59	F
PUBLICATION_FOR_SAVED_CONFIG, page 17-61	F
REGISTER_MODEL_TO_POOL, page 17-63	P
UNREGISTER_MODEL_FROM_POOL, page 17-64	P
UNREGISTER_POOL, page 17-65	P
UI_FOR_ITEM, page 17-66	F
UI_FOR_PUBLICATION_ID, page 17-68	F
VALIDATE, page 17-69	P
CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION, page 17-72	P

COMMON_BILL_FOR_ITEM

This procedure retrieves the common bill item, if any, for the organization ID and inventory item ID that are passed in as parameters.

This procedure is used by the PUBLICATION_FOR_ITEM, page 17-58 function to retrieve the common bill's details if the Model has not been published.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE common_bill_for_item ( in_inventory_item_id IN NUMBER,
                                 in_organization_id IN NUMBER,
                                 common_inventory_item_id OUT NOCOPY
NUMBER,
                                 common_organization_id OUT NOCOPY
NUMBER);
```

The table Parameters for the COMMON_BILL_FOR_ITEM Procedure, page 17-15 lists the parameters for the COMMON_BILL_FOR_ITEM procedure. The description includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the COMMON_BILL_FOR_ITEM Procedure

Parameter	Data Type	Mode	Note
in_inventory_item_id	number	in	Inventory Item ID of item for which common bill may be defined.
in_organization_id	number	in	Organization ID of Item for which common bill may be defined.
common_inventory_item_id	number	out	Inventory Item ID of the common bill item. NULL if no common bill defined.
common_organization_id	number	out	Organization ID of the common bill Item. NULL if no common bill defined.

CONFIG_MODEL_FOR_ITEM

This function finds a published configuration model for an item, and other applicability parameters. Returns NULL if the Model cannot be found.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not

defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_model_for_item (inventory_item_id IN NUMBER,
                                organization_id IN NUMBER,
                                config_lookup_date IN DATE,
                                calling_application_id IN NUMBER,
                                usage_name IN VARCHAR2,
                                publication_mode IN VARCHAR2 DEFAULT
NULL,
                                language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The table Parameters for the CONFIG_MODEL_FOR_ITEM Function, page 17-16 describes the parameters for the CONFIG_MODEL_FOR_ITEM function. The table includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_MODEL_FOR_ITEM Function

Parameter	Data Type	Mode	Note
inventory_item_id	number	in	If the Model was imported from Oracle BOM, this is the Inventory Item ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number	in	If the Model was imported from Oracle BOM, this is the organization ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the `devl_project_id` of the configuration model published for this combination of inputs. NULL is returned if there is no matching publication.

CONFIG_MODELS_FOR_ITEMS

This function finds the Models that are associated with each entry in a list of Inventory Items that are published with the matching applicability parameters. The function returns the list of Model IDs (`devl_project_id` values) that meet the specified parameters.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be

checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_models_for_items (inventory_item_id IN NUMBER_TBL_TYPE,
                                  organization_id IN NUMBER_TBL_TYPE,
                                  config_lookup_date IN DATE_TBL_TYPE,
                                  calling_application_id IN
NUMBER_TBL_TYPE,
                                  usage_name IN VARCHAR2_TBL_TYPE,
                                  publication_mode IN VARCHAR2_TBL_TYPE,
                                  language IN VARCHAR2_TBL_TYPE)
RETURN NUMBER_TBL_TYPE;
```

The following table describes the parameters for the CONFIG_MODELS_FOR_ITEMS function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_MODELS_FOR_ITEMS Function

Parameter	Data Type	Mode	Note
inventory_item_id	number_tbl_type	in	If the Model was imported from Oracle BOM, this is a list of Inventory Item IDs for the published Model from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number_tbl_type	in	If the Model was imported from Oracle BOM, this is a list of organization IDs for the published Model from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_lookup_date	date_tbl_type	in	List of dates to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
calling_application_id	number_tbl_type	in	List of registered IDs of applications for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2_tbl_type	in	List of Usage names to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2_tbl_type	in	List of publication modes to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2_tbl_type	in	List of language codes to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns an array in which each element is a `devl_project_id` value for the associated item. NULL is returned if there is no matching publication.

CONFIG_MODEL_FOR_PRODUCT

This function finds a published configuration model for a product key and other applicability parameters. Returns NULL if the Model cannot be found.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_model_for_product (product_key IN VARCHAR2,
                                   config_lookup_date IN DATE,
                                   calling_application_id IN NUMBER,
                                   usage_name IN VARCHAR2,
                                   publication_mode IN VARCHAR2 DEFAULT
NULL,
                                   language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the CONFIG_MODEL_FOR_PRODUCT function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_MODEL_FOR_PRODUCT Function

Parameter	Data Type	Mode	Note
product_key	varchar2	in	Product key to search for in the publication. See Applicability Parameters, page 17-9.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the `devl_project_id` of the configuration model published for this combination of inputs. NULL is returned if there is no matching publication.

CONFIG_MODELS_FOR_PRODUCTS

This function returns a list of Model IDs (`devl_project_id` values) associated with each entry in a list of product keys that are published with matching applicability parameters.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_models_for_products ( product_key IN VARCHAR2_TBL_TYPE,
                                      config_lookup_date IN
DATE_TBL_TYPE,
                                      calling_application_id IN
NUMBER_TBL_TYPE,
                                      usage_name IN VARCHAR2_TBL_TYPE,
                                      publication_mode IN
VARCHAR2_TBL_TYPE,
                                      language IN VARCHAR2_TBL_TYPE)
RETURN NUMBER_TBL_TYPE;
```

The following table describes the parameters for the CONFIG_MODELS_FOR_PRODUCTS function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_MODELS_FOR_PRODUCTS Function

Parameter	Data Type	Mode	Note
product_key	varchar2_tbl_type	in	List of product keys to search for in the publication. See Applicability Parameters, page 17-9.
config_lookup_date	date_tbl_type	in	List of dates to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
calling_application_id	number_tbl_type	in	List of registered IDs of applications for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2_tbl_type	in	List of Usage names to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2_tbl_type	in	List of publication modes to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2_tbl_type	in	List of language codes to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns a list of Model IDs (`devl_project_id` values) associated with each entry in a list of product keys that are published with matching applicability parameters.

CONFIG_UI_FOR_ITEM

This function returns the user interface ID associated with the publication found for the input item, organization ID, and applicability.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_ui_for_item (inventory_item_id IN NUMBER,
                             organization_id IN NUMBER,
                             config_lookup_date IN DATE,
                             ui_type IN OUT NOCOPY VARCHAR2,
                             calling_application_id IN NUMBER,
                             usage_name IN VARCHAR2,
                             publication_mode IN VARCHAR2 DEFAULT NULL,
                             language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the CONFIG_UI_FOR_ITEM function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_UI_FOR_ITEM Function

Parameter	Data Type	Mode	Note
inventory_item_id	number	in	If the Model was imported from Oracle BOM, this is the Inventory Item ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number	in	If the Model was imported from Oracle BOM, this is the organization ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.

Parameter	Data Type	Mode	Note
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.
ui_type	varchar2	in/out	This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'. If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD. If DHTML or JRAD is passed and there is no publication available for the item, then the **API** returns the user interface ID of the BOM JRAD UI.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the user interface ID associated with the selected publication.

If the `ui_type` is APPLET, then the publication UI type can be either APPLET, DHTML, or JRAD.

If the `ui_type` is either DHTML or JRAD, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.

CONFIG_UI_FOR_ITEM_LF

This function does the same work as CONFIG_UI_FOR_ITEM, page 17-23, but also returns the look_and_feel of the UI ('APPLET', 'BLAF', or 'FORMS').

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_ui_for_item_lf ( inventory_item_id IN NUMBER,
                                 organization_id IN NUMBER,
                                 config_lookup_date IN DATE,
                                 ui_type IN OUT NOCOPY VARCHAR2,
                                 calling_application_id IN NUMBER,
                                 usage_name IN VARCHAR2,
                                 look_and_feel OUT NOCOPY VARCHAR2,
                                 publication_mode IN VARCHAR2 DEFAULT
NULL,
                                 language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the CONFIG_UI_FOR_ITEM_LF function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_UI_FOR_ITEM_LF Function

Parameter	Data Type	Mode	Note
inventory_item_id	number	in	If the Model was imported from Oracle BOM, this is the Inventory Item ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number	in	If the Model was imported from Oracle BOM, this is the organization ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
ui_type	varchar2	in/out	This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'. If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD. If DHTML or JRAD is passed and there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.
look_and_feel	varchar2	out	This is a tag that overrides the default look and feel for component-style UIs (when UI_STYLE=0) in the CZ_UI_DEFS table.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the user interface ID associated with the selected publication.

If the `ui_type` is APPLET, then the publication UI type can be either APPLET, DHTML, or JRAD.

If the `ui_type` is either DHTML or JRAD, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.

CONFIG_UI_FOR_PRODUCT

This function finds a UI for a product, and returns NULL if no UI can be found. If ui_type is passed in, the function will validate the UI it finds against this type. If the types do not match, no UI will be returned. If no ui_type is passed, the type of the UI will be returned in ui_type.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_ui_for_product ( product_key IN VARCHAR2,
                                 config_lookup_date IN DATE,
                                 ui_type IN OUT NOCOPY VARCHAR2,
                                 calling_application_id IN NUMBER,
                                 usage_name IN VARCHAR2,
                                 publication_mode IN VARCHAR2 DEFAULT
NULL,
                                 language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the CONFIG_UI_FOR_PRODUCT function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_UI_FOR_PRODUCT Function

Parameter	Data Type	Mode	Note
product_key	varchar2	in	Product key to search for in the publication. See Applicability Parameters, page 17-9.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.
ui_type	varchar2	in/out	This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'. If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD. If DHTML or JRAD is passed and there is no publication available for the item, and if the product_key corresponds to the inventory item, then the user interface ID of the BOM UI is returned

Parameter	Data Type	Mode	Note
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

If `ui_type` is passed in, then the function will validate the UI it finds against this type. This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'.

If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If DHTML or JRAD is passed and the item does not have a publication available, and if the `product_key` corresponds to the inventory item, then the user interface ID of the BOM UI is returned.

If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD.

CONFIG_UIS_FOR_ITEMS

This function returns a list of user interfaces that are associated with each entry in the list of Inventory Items that are published with matching applicability parameters.

Considerations Before Running

None

Timing

This function should be used after publishing Models to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_uis_for_items ( inventory_item_id IN NUMBER_TBL_TYPE,
                                organization_id IN NUMBER_TBL_TYPE,
                                config_lookup_date IN DATE_TBL_TYPE,
                                ui_type IN OUT NOCOPY VARCHAR2_TBL_TYPE,
                                calling_application_id IN
NUMBER_TBL_TYPE,
                                usage_name IN VARCHAR2_TBL_TYPE,
                                publication_mode IN VARCHAR2_TBL_TYPE,
                                language IN VARCHAR2_TBL_TYPE )
RETURN NUMBER_TBL_TYPE;
```

The following table describes the parameters for the CONFIG_UIS_FOR_ITEMS function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_UIS_FOR_ITEMS Function

Parameter	Data Type	Mode	Note
inventory_item_id	number_tbl_type	in	If the Model was imported from Oracle BOM, this is a list of Inventory Item IDs for the published Model from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number_tbl_type	in	If the Model was imported from Oracle BOM, this is a list of organization IDs for the published Model from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_lookup_date	date_tbl_type	in	List of dates to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.
ui_type	varchar2_tbl_type	in/ out	This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'. If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD. If DHTML or JRAD is passed and there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.
calling_application_id	number_tbl_type	in	List of registered IDs of applications for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2_tbl_type	in	List of Usage names to search for in the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
publication_mode	varchar2_tbl_type	in	List of publication modes to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2_tbl_type	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the user interface ID associated with the selected publication.

If the `ui_type` is APPLET, then the publication UI type can be either APPLET, DHTML, or JRAD.

If the `ui_type` is either DHTML or JRAD, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.

CONFIG_UIS_FOR_PRODUCTS

This function returns a list of user interfaces that are associated with each entry in the list of product keys that are published with matching applicability parameters.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION config_uis_for_products ( product_key IN VARCHAR2_TBL_TYPE,
                                   config_lookup_date IN DATE_TBL_TYPE,
                                   ui_type IN OUT NOCOPY
VARCHAR2_TBL_TYPE,
                                   calling_application_id IN
NUMBER_TBL_TYPE,
                                   usage_name IN VARCHAR2_TBL_TYPE,
                                   publication_mode IN VARCHAR2_TBL_TYPE,
                                   language IN VARCHAR2_TBL_TYPE )
RETURN NUMBER_TBL_TYPE;
```

The following table describes the parameters for the CONFIG_UIS_FOR_PRODUCTS function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CONFIG_UIS_FOR_PRODUCTS Function

Parameter	Data Type	Mode	Note
product_key	varchar2_tbl_type,	in	List of product keys to search for in the publication. See Applicability Parameters, page 17-9.
config_lookup_date	date_tbl_type,	in	List of dates to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
ui_type	varchar2_tbl_type,	in/out	This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'. If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD. If DHTML or JRAD is passed and there is no publication available for the item, and if the product_key corresponds to the inventory item, then the user interface ID of the BOM UI is returned
calling_application_id	number_tbl_type,	in	List of registered IDs of applications for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2_tbl_type,	in	List of Usage names to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2_tbl_type,	in	List of publication modes to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2_tbl_type	in	List of language codes to search for in the publication. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

If `ui_type` is passed in, then the function will validate the UI it finds against this type. This is the type of published UI sought and found for each product. Values are

'APPLET', 'DHTML', or 'JRAD'.

If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If DHTML or JRAD is passed and the item does not have a publication available, and if the `product_key` corresponds to the inventory item, then the user interface ID of the BOM UI is returned.

If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD.

COPY_CONFIGURATION

This procedure in the CZ_CF_API package is used to copy configurations models. It is not to be used to copy networked configuration models.

This procedure copies a configuration in the database. If the NEW_CONFIG_FLAG is 1, then a new CONFIG_HDR_ID value is generated for the new configuration and it is REV_NBR 1. If NEW_CONFIG_FLAG is 0, the copy keeps the CONFIG_HDR_ID and has a REV_NBR incremented to be greater than the original.

Considerations Before Running

None

Prerequisites

The configuration to be copied must exist. This procedure must not be used with networked Models.

> **Note:** If you want to copy a networked configuration model, then you must use the copy_configuration procedure in the CZ_CONFIG_API_PUB package. For more information see CZ_CONFIG_API_PUB.COPY_CONFIGURATION, page 17-39.

Timing

This procedure should be used every time a configuration is copied. The procedure will ensure that all inputs, outputs, attributes, and messages are copied.

Warnings

If the configuration does not exist, or if the copy fails, return_value will be zero, and error_message will contain error information.

> **Note:** COPY_CONFIGURATION procedure does not commit the copy data. It is your responsibility to commit the copied configuration.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE copy_configuration( config_hdr_id       IN  NUMBER,
                              config_rev_nbr      IN  NUMBER,
                              new_config_flag     IN  VARCHAR2,
                              out_config_hdr_id   IN OUT NOCOPY NUMBER,
                              out_config_rev_nbr  IN OUT NOCOPY NUMBER,
                              error_message       IN OUT NOCOPY
VARCHAR2,
                              return_value        IN OUT NOCOPY NUMBER,
                              handle_deleted_flag IN  VARCHAR2 DEFAULT
NULL,
                              new_name            IN  VARCHAR2 DEFAULT
NULL);
```

The following table describes the parameters for the COPY_CONFIGURATION procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the COPY_CONFIGURATION Procedure

Parameter	Data Type	Mode	Note
config_hdr_id	number	in	Specifies which configuration to copy. Uses CZ_CONFIG_HDRS, CZ_CONFIG_INPUTS, CZ_CONFIG_ITEMS, CZ_CONFIG_MESSAGES, and CZ_CONFIG_ATTRIBUTES.
config_rev_nbr	number	in	Specifies which configuration to copy. Uses CZ_CONFIG_HDRS, CZ_CONFIG_INPUTS, CZ_CONFIG_ITEMS, CZ_CONFIG_MESSAGESl, and CZ_CONFIG_ATTRIBUTES.
new_config_flag	varchar2	in	A '1' indicates that the copied configuration should have a new CONFIG_HDR_ID. A '0' indicates that the copied configuration should have the same CONFIG_HDR_ID and a unique CONFIG_REV_NBR. For example it is a revision of the existing configuration.

Parameter	Data Type	Mode	Note
out_config_hdr_id	number	in/out	Identifies the new copy of the configuration.
out_config_rev_nbr	number	in/out	Identifies the new copy of the configuration.
error_message	varchar2	in/out	Contains an error message if an error occurs.
return_value	number	in/out	Indicates the success (1) or failure (0) of the copy.
handle_deleted_flag	varchar2	in	When '0', it will undelete the copied configuration if the original configuration is deleted.
new_name	varchar2	in	Applies a new name for the configuration

Considerations After Running

None

Results

This procedure copies all database records associated with a configuration to a new config_hdr_id and config_rev_nbr.

Troubleshooting

Examine return_value and error_message to determine what the next step should be

CZ_CONFIG_API_PUB.COPY_CONFIGURATION

This API procedure in the CZ_CONFIG_API_PUB package is used to copy configurations as well as configurations that contain connectors and support connectivity.

This procedure creates a new configuration by copying the original configuration's CONFIG_HDR_ID and CONFIG_REV_NBR

This procedure copies a configuration in the database. If the NEW_CONFIG_FLAG is 1, then a new CONFIG_HDR_ID value is generated for the new configuration and it is REV_NBR 1. If NEW_CONFIG_FLAG is 0, the copy keeps the CONFIG_HDR_ID and

has a REV_NBR incremented to be greater than the original.

Considerations Before Running

None

Prerequisites

The configuration to be copied must exist.

Timing

This procedure should be used every time a configuration is copied. The procedure will ensure that all inputs, outputs, attributes, and messages are copied.

Warnings

If the configuration does not exist, or if the copy fails, return_status will be FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR if an error occurs within the procedure, and msg_data will contain error information.

> **Note:** CZ_CONFIG_API_PUB.COPY_CONFIGURATION procedure does not commit the copy data. It is your responsibility to commit the copied configuration.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE copy_configuration(p_api_version          IN  NUMBER,
                             p_config_hdr_id        IN  NUMBER,
                             p_config_rev_nbr       IN  NUMBER,
                             p_copy_mode            IN  VARCHAR2,
                             x_config_hdr_id        OUT NOCOPY  NUMBER,
                             x_config_rev_nbr       OUT NOCOPY  NUMBER,
                             x_orig_item_id_tbl     OUT NOCOPY
CZ_API_PUB.number_tbl_type,
                             x_new_item_id_tbl      OUT NOCOPY
CZ_API_PUB.number_tbl_type,
                             x_return_status        OUT NOCOPY
VARCHAR2,
                             x_msg_count            OUT NOCOPY  NUMBER,
                             x_msg_data             OUT NOCOPY
VARCHAR2,
                             p_handle_deleted_flag  IN  VARCHAR2 :=
NULL,
                             p_new_name             IN  VARCHAR2 :=
NULL);
```

The following table describes the parameters for the CZ_CONFIG_API_PUB.COPY_CONFIGURATION procedure. This includes the data

type, the mode (in or out), and a brief note about the parameter.

Parameters for the CZ_CONFIG_API_PUB.COPY_CONFIGURATION Procedure

Parameter	Data Type	Mode	Note
p_api_version	number	in	Required. See API Version Numbers, page 18-9
p_config_hdr_id	number	in	Required. Specifies which configuration to copy. Uses CZ_CONFIG_HDRS, CZ_CONFIG_INPUTS, CZ_CONFIG_ITEMS, CZ_CONFIG_MESSAGES, and CZ_CONFIG_ATTRIBUTES.
p_config_rev_nbr	number	in	Required. Specifies which configuration to copy. Uses CZ_CONFIG_HDRS, CZ_CONFIG_INPUTS, CZ_CONFIG_ITEMS, CZ_CONFIG_MESSAGESl, and CZ_CONFIG_ATTRIBUTES.
x_config_hdr_id	number	out	Identifies the new copy of the configuration.
x_config_rev_nbr	number	out	Identifies the new copy of the configuration.
p_copy_mode	varchar2	in	Required. Specifies whether the new configuration has a new header ID or a new revision number.
x_orig_item_id_tbl	number	out	A table of the item IDs for the items in the original configuration.
x_new_item_id_tbl	number	out	A table of the item IDS for the items in the new configuration.
x_return_status	varchar2	out	Must return FND_API.G_RET_STS_SUCCESS if procedure completed successfully; otherwise return FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR if an error occurs within the procedure

Parameter	Data Type	Mode	Note
x_msg_count	number	out	Required. The number of error messages returned in the `x_msg_data` parameter.
x_msg_data	varchar2	out	Contains an error message if the procedure is returning an `x_return_status` value of FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR
p_handle_deleted_flag	varchar2	in	When '0', it will undelete the copied configuration if the original configuration is deleted.
p_new_name	varchar2	in	Applies a new name for the configuration

COPY_CONFIGURATION_AUTO

This procedure runs COPY_CONFIGURATION, page 17-37 within an autonomous transaction. If the copy is successful, new data will be committed to the database without affecting the caller's transaction.

See other information for COPY_CONFIGURATION, page 17-37.

Considerations Before Running

None

Prerequisites

The configuration to be copied must exist. This procedure must not be used with networked Models.

> **Note:** If you want to copy a networked configuration model autonomously, then you must use the copy_configuration procedure in the CZ_CONFIG_API_PUB package. For more information see CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO, page 17-45.

Timing

This procedure should be used every time a configuration is copied. The procedure will ensure that all inputs, outputs, attributes, and messages are copied.

Warnings

If the configuration does not exist, or if the copy fails, return_value will be zero, and error_message will contain error information.

> **Note:** COPY_CONFIGURATION_AUTO procedure does not commit the copy data. It is your responsibility to commit the copied configuration.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE copy_configuration_auto(config_hdr_id       IN  NUMBER,
                                  config_rev_nbr      IN  NUMBER,
                                  new_config_flag     IN  VARCHAR2,
                                  out_config_hdr_id   IN OUT NOCOPY
NUMBER,
                                  out_config_rev_nbr  IN OUT NOCOPY
NUMBER,
                                  Error_message       IN OUT NOCOPY
VARCHAR2,
                                  Return_value        IN OUT NOCOPY
NUMBER,
                                  handle_deleted_flag IN  VARCHAR2
DEFAULT NULL,
                                  new_name           IN  VARCHAR2
DEFAULT NULL);
```

The following table describes the parameters for the COPY_CONFIGURATION_AUTO procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the COPY_CONFIGURATION_AUTO Procedure

Parameter	Data Type	Mode	Note
config_hdr_id	number	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
config_rev_nbr	number	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.

Parameter	Data Type	Mode	Note
new_config_flag	varchar2	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
out_config_hdr_id	number	in/out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
out_config_rev_nbr	number	in/out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
error_message	varchar2	in/out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
return_value	number	in/out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
handle_deleted_flag	varchar2 default null	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
new_name	varchar2 default null	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.

Considerations After Running

None

Results

This procedure copies all database records associated with a configuration to a new config_hdr_id and config_rev_nbr.

Troubleshooting

Examine return_value and error_message to determine what the next step should be.

CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO

This procedure runs COPY_CONFIGURATION, page 17-37 within an autonomous transaction. If the copy is successful, new data will be committed to the database without affecting the caller's transaction. This procedure can be used with networked configurations.

See other information for COPY_CONFIGURATION, page 17-37.

Considerations Before Running

None

Prerequisites

The configuration to be copied must exist.

Timing

This procedure should be used every time a configuration is copied. The procedure will ensure that all inputs, outputs, attributes, and messages are copied.

Warnings

If the configuration does not exist, or if the copy fails, return_status will be FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR if an error occurs within the procedure, and msg_data will contain error information.

> **Note:** CZ_AUTO_API_PUB.COPY_CONFIGURATION_AUTO procedure does not commit the copy data. It is your responsibility to commit the copied configuration.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE copy_configuration_auto ( p_api_version          IN  NUMBER,
        p_config_hdr_id        IN  NUMBER,
        p_config_rev_nbr       IN  NUMBER,
        p_copy_mode            IN  VARCHAR2,
        x_config_hdr_id        OUT NOCOPY  NUMBER,
        x_config_rev_nbr       OUT NOCOPY  NUMBER,
        x_orig_item_id_tbl     OUT NOCOPY  CZ_API_PUB.number_tbl_type,
        x_new_item_id_tbl      OUT NOCOPY  CZ_API_PUB.number_tbl_type,
        x_return_status        OUT NOCOPY  VARCHAR2,
        x_msg_count            OUT NOCOPY  NUMBER,
        x_msg_data             OUT NOCOPY  VARCHAR2,
        p_handle_deleted_flag  IN  VARCHAR2 := NULL,
        p_new_name             IN  VARCHAR2 := NULL);
```

The following table describes the parameters for the CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the CZ_CONFIG_API_PUB.COPY_CONFIGURATION_AUTO Procedure

Parameter	Data Type	Mode	Note
p_api_version	number	in	See API Version Numbers, page 18-9.
p_config_hdr_id	number	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
p_config_rev_nbr	number	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
p_copy_mode	varchar2	in	Required. Specifies whether the new configuration has a new header ID or a new revision number.
x_config_hdr_id	number	out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
x_config_rev_nbr	number	out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.

Parameter	Data Type	Mode	Note
x_orig_item_id_tbl	number	out	A table of the item IDs for the items in the original configuration.
x_new_item_id_tbl	number	out	A table of the item IDS for the items in the new configuration.
x_msg_count	number	out	Required. The number of error messages returned in the `x_msg_data` parameter.
x_msg_data	varchar2	out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
x_return_status	number	out	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
p_handle_deleted_flag	varchar2 default null	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.
p_new_name	varchar2 default null	in	See corresponding parameter in Parameters for the COPY_CONFIGURATION Procedure, page 17-38.

Considerations After Running

None

Results

This procedure copies all database records associated with a configuration to a new config_hdr_id and config_rev_nbr.

Troubleshooting

Examine return_value and error_message to determine what the next step should be.

DEFAULT_NEW_CFG_DATES

This utility procedure provides default date values used by Oracle Configurator for a new configuration. The caller should pass in dates that will be included in the initialization message for the runtime Oracle Configurator. The procedure will return the value that will be used by the runtime Oracle Configurator for any dates not passed in.

Considerations Before Running

None

Prerequisites

None.

Timing

This procedure should be used to find out the default dates used by the runtime Oracle Configurator for publication lookup, effectivity, and configuration creation.

Dependencies

None.

Restrictions and Limitations

None.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE DEFAULT_NEW_CFG_DATES( p_creation_date IN OUT NOCOPY DATE,
                                 p_lookup_date IN OUT NOCOPY DATE,
                                 p_effective_date IN OUT NOCOPY DATE);
```

The following table describes the parameters for the DEFAULT_NEW_CFG_DATES procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the DEFAULT_NEW_CFG_DATES Procedure

Parameter	Data Type	Mode	Note
p_creation_date	date	in/out	This specifies the creation date for the new configuration.
p_lookup_date	date	in/out	This specifies the lookup date for the new configuration.
p_effective_date	date	in/out	This specifies the effective date for the new configuration.

Considerations After Running

None

Results

Any of the parameters (`p_creation_date`, `p_lookup_date`, `p_effective_date`) that were not passed in are populated with the date that the runtime Oracle Configurator would use for that parameter.

DEFAULT_RESTORED_CFG_DATES

This utility procedure provides default date values used by Oracle Configurator for a restored configuration. The caller should pass in dates that will be included in the initialization message for the runtime Oracle Configurator. The procedure will return the value that will be used by the runtime Oracle Configurator for any dates not passed in. The CONFIG_HEADER_ID and a configuration revision (CONFIG_REV_NBR) must be supplied. .

Considerations Before Running

None

Prerequisites

Configuration must exist.

Timing

This procedure should be used to find out the default dates used by the runtime Oracle Configurator for publication lookup, effectivity, and configuration creation.

Dependencies

None.

Restrictions and Limitations

None.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE DEFAULT_RESTORED_CFG_DATES( p_config_hdr_id IN NUMBER,
                                      p_config_rev_nbr IN NUMBER,
                                      p_creation_date IN OUT NOCOPY
DATE,
                                      p_lookup_date IN OUT NOCOPY DATE,
                                      p_effective_date IN OUT NOCOPY
DATE );
```

The following table describes the parameters for the DEFAULT_RESTORED_CFG_DATES procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the DEFAULT_RESTORED_CFG_DATES Procedure

Parameter	Data Type	Mode	Note
p_config_hdr_id	number	in	Specifies which configuration to use.
p_config_rev_nbr	number	in	Specifies which configuration to use
p_creation_date	date	in/out	If this is not null, then it will be returned as is. If this is null and if `p_lookup_date` is null and RestoredConfigDefaultModelLookupDate in CZ_DB_SETTINGS is set to `config_creation_date`, then `sysdate` is returned. See RestoredConfigDefaultModelLookupDate, page 4-23 for more information

Parameter	Data Type	Mode	Note
p_lookup_date	date	in/out	If this is not null, then it will be returned as is. If this is null, and if RestoredConfigDefaultModelLookupDate in CZ_DB_SETTINGS is set to `config_creation_date`, then `p_lookup_date` is set to the order line creation date. If RestoredConfigDefaultModelLookupDate in CZ_DB_SETTINGS is not set to `config_creation_date`, then `sysdate` is returned. See RestoredConfigDefaultModelLookupDate, page 4-23 for more information.
p_effective_date	date	in/out	If this is not null, then it will be returned as is. Otherwise, the existing setting for this configuration is returned.

Considerations After Running

None

Results

Any of the parameters (`p_creation_date`, `p_lookup_date`, `p_effective_date`) that were not passed in are populated with the date that the runtime Oracle Configurator would use for that parameter.

DELETE_CONFIGURATION

This procedure removes a configuration from the database.

Considerations Before Running

None

Prerequisites

The configuration to be deleted must exist. If the specified configuration does not exist, then the procedure runs but it does not delete anything and no issues are reported.

Timing

This procedure should be used when a configuration is obsolete.

Warnings

Do not delete configurations that are referred to by any host applications.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE delete_configuration( config_hdr_id  IN  NUMBER,
                                config_rev_nbr IN  NUMBER,
                                usage_exists   IN OUT NOCOPY NUMBER,
                                Error_message  IN OUT NOCOPY VARCHAR2,
                                Return_value   IN OUT NOCOPY NUMBER);
```

The following table describes the parameters for the DELETE_CONFIGURATION procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the DELETE_CONFIGURATION Procedure

Parameter	Data Type	Mode	Note
config_hdr_id	number	in	Specifies the header ID of the configuration to be deleted
config_rev_nbr	number	in	Specifies the revision number of the configuration to be deleted
usage_exists	number	in/out	This returns 1 if a configuration usage record exists and the configuration is not deleted. (Requires custom code to populate the CZ_CONFIG_USAGES table.)
error_message	varchar2	in/out	If there is an error, then this field contains a message describing the error.
return_value	number	in/out	If 1, then the configuration was successfully deleted. If 0, then deletion of the configuration failed.

Considerations After Running

None

Troubleshooting

Examine the output in the `error_message` parameter.

ICX_SESSION_TICKET

This function returns a value for the ICX session ticket that Oracle Applications should pass in the `icx_session_ticket` parameter of the initialization message when calling Oracle Configurator. See icx_session_ticket, page 9-26 in Session Initialization, page 9-1 for information about that parameter.

The session ticket allows the runtime Oracle Configurator to maintain the Oracle Applications session identity. A null value is returned if `user_id`, `resp_id`, or `appl_id` are not defined within the Oracle Applications session or if the ICX calls fail.

For more information about the ICX session ticket, including the profile option ICX: Session Timeout, see the *Oracle E-Business Suite System Administrator's Guide - Maintenance*.

Considerations Before Running

None

Prerequisites

In order to use this function, the database session must have been initialized with Oracle Applications parameters in order for the icx_session_ticket to return a value.

Timing

This function should be used before launching a configuration session from PL/SQL.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION icx_session_ticket RETURN VARCHAR2;
```

There are no parameters for this function. It derives its inputs from the environment of the database session.

Considerations After Running

None

Results

This function returns the ICX ticket that represents the Oracle Applications session.

Troubleshooting

If this function returns NULL, the database session is not an Oracle Applications session.

MODEL_FOR_ITEM

This function returns a published Model passed on the inventory item ID, organization id, and applicability.

This function is used for backward compatibility. It calls CONFIG_MODEL_FOR_ITEM, page 17-15 with usage_name equal to "Any Usage" and publication_mode equal to 'P'.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION model_for_item( inventory_item_id NUMBER,
                         organization_id NUMBER,
                         config_creation_date DATE,
                         user_id NUMBER,
                         responsibility_id NUMBER,
                         calling_application_id NUMBER )
RETURN NUMBER;
```

The following table lists of the parameters for the MODEL_FOR_ITEM function, including the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the MODEL_FOR_ITEM Function

Parameter	Data Type	Mode	Note
inventory_item_id	number	in	If the Model was imported from Oracle BOM, then this is the inventory item ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number	in	If the Model was imported from Oracle BOM, then this is the organization ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_creation_date	date	in	This is the lookup date for the configuration
user_id	number	in	This is the ID for the Oracle Applications user that is logged into from FND_USER.
responsibility_id	number	in	This is the responsibility that the Oracle Applications user had in the host application.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the `devl_project_id` of the configuration model published for this combination of inputs. NULL is returned if there is no matching publication.

MODEL_FOR_PUBLICATION_ID

This function returns the Model ID for a specified publication.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION model_for_publication_id (publication_id NUMBER)
RETURN NUMBER;
```

The following table describes the parameters for the MODEL_FOR_PUBLICATION_ID function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the MODEL_FOR_PUBLICATION_ID Function

Parameter	Data Type	Mode	Note
publication_id	number	in	This is the specified publication id in the CZ_MODEL_PUBLICATIONS table.

POOL_TOKEN_FOR_PRODUCT_KEY

This function returns the name of the JVM pool registered for a given Product Key, by looking up the JVM pool registered to the specified Model in the mapping table (CZ_MODEL_POOL_MAPPINGS).

Considerations Before Running

Use of this function assumes that you are routing Models to JVM pools, as described in Routing Models to Specified JVMs, page 20-10.

Timing

This function should be used when you need to obtain the name of the JVM pool to which a specific Model is registered. The Model is identified by the Product Key.

Dependencies

The profile option CZ: Add Model Routing Cookie must be set to True for Model routing to occur at runtime.

Syntax and Parameters

The syntax for this procedure is:

Example

```
FUNCTION pool_token_for_product_key (p_product_key IN VARCHAR2)
RETURN VARCHAR2;
```

The following table describes the parameters for the POOL_TOKEN_FOR_PRODUCT_KEY function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the POOL_TOKEN_FOR_PRODUCT_KEY Function

Parameter	Data Type	Mode	Note
p_product_key	varchar2	in	Product Key of the Model for which the registered pool name is desired. For details on Product Key, see Applicability Parameters, page 17-9.

PUBLICATION_FOR_ITEM

This function returns the publication ID for a specified inventory item.

Considerations Before Running

None

Timing

This function should be used after publishing Models to verify that publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION publication_for_item ( inventory_item_id IN NUMBER,
                                organization_id IN NUMBER,
                                config_lookup_date IN DATE,
                                calling_application_id IN NUMBER,
                                usage_name IN VARCHAR2,
                                publication_mode IN VARCHAR2 DEFAULT
NULL,
                                language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the PUBLICATION_FOR_ITEM function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the PUBLICATION_FOR_ITEM Function

Parameter	Data Type	Mode	Note
inventory_item_id	number	in	If the Model was imported from Oracle BOM, then this is the Inventory Item ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number	in	If the Model was imported from Oracle BOM, then this is the organization ID for the published Model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

PUBLICATION_FOR_PRODUCT

This function returns the publication ID for a product key.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a Model to be returned. This function must be run on the instance that the Model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION publication_for_product( product_key IN VARCHAR2,
                                  config_lookup_date IN DATE,
                                  calling_application_id IN NUMBER,
                                  usage_name IN VARCHAR2,
                                  publication_mode IN VARCHAR2 DEFAULT
NULL,
                                  language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the PUBLICATION_FOR_PRODUCT function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the PUBLICATION_FOR_PRODUCT Function

Parameter	Data Type	Mode	Note
product_key	varchar2	in	Product key to search for in the publication. See Applicability Parameters, page 17-9.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Applicability Parameters, page 17-9.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

PUBLICATION_FOR_SAVED_CONFIG

This function is used to determine the publication that should be used to reopen a saved configuration. The function returns a publication ID for an existing configuration based on its model information and applicability parameters.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a model to be returned. This function must be run on the instance that the model is published to.

Warnings

If `usage_name` and/or `publication_mode` are NULL or not provided, the CZ: Publication Usage and/or CZ: Publication Lookup Mode profile option values will be checked. However, Oracle Applications session parameters are not defined by default within a SQL*Plus session. If profile option values are not defined for this or any other reason, the defaults for `usage_name` and/or `publication_mode` will be "Any Usage" and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION publication_for_saved_config ( config_hdr_id IN NUMBER,
                                        config_rev_nbr IN NUMBER,
                                        config_lookup_date IN DATE,
                                        calling_application_id IN
NUMBER,
                                        usage_name IN VARCHAR2,
                                      publication_mode IN VARCHAR2
DEFAULT NULL,
                                      language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

The following table describes the parameters for the PUBLICATION_FOR_SAVED_CONFIG function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the PUBLICATION_FOR_SAVED_CONFIG Function

Parameter	Data Type	Mode	Note
config_hdr_id	number	in	Identifies the saved configuration to use.
config_rev_nbr	number	in	Identifies the saved configuration.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Applicability Parameters, page 17-9.

Parameter	Data Type	Mode	Note
calling_application_id	number	in	The registered ID of an application for which the model is published. See Applicability Parameters, page 17-9.
usage_name	varchar2	in	Usage name to search for in the publication. See Applicability Parameters, page 17-9.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Applicability Parameters, page 17-9.
language	varchar2	in	Language code to search for in the publication. See Applicability Parameters, page 17-9.

REGISTER_MODEL_TO_POOL

This procedure registers a Model to a JVM pool, by creating a mapping in the mapping table (CZ_MODEL_POOL_MAPPINGS) that registers the specified Model to the specified JVM pool.

If references to the specified pool do not exist in the mapping table, this procedure creates rows that implicitly register that pool, with an autonomous transaction.

Considerations Before Running

Use of this procedure assumes that you are routing Models to JVM pools, as described in Routing Models to Specified JVMs, page 20-10.

Timing

This procedure should be used when you need to register a Model to a JVM to reduce the Model's memory footprint and improve performance.

Dependencies

The profile option CZ: Add Model Routing Cookie must be set to True for Model routing to occur at runtime.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE register_model_to_pool (p_pool_identifier IN VARCHAR2,
                                  p_model_product_key IN VARCHAR2);
```

The following table describes the parameters for the REGISTER_MODEL_TO_POOL function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the REGISTER_MODEL_TO_POOL Function

Parameter	Data Type	Mode	Note
p_pool_identifier	varchar2	in	The JVM pool to which the specified Model is to be registered.
p_model_product_key	varchar2	in	Product Key of the Model to be registered. For details on Product Key, see Applicability Parameters, page 17-9.

UNREGISTER_MODEL_FROM_POOL

This procedure unregisters a Model from a JVM pool, by deleting the mapping in the mapping table (CZ_MODEL_POOL_MAPPINGS) that registered the specified Model to the specified JVM pool. Uses an autonomous transaction.

Considerations Before Running

Use of this procedure assumes that you are routing Models to JVM pools, as described in Routing Models to Specified JVMs, page 20-10.

Timing

This procedure should be used when you need to unregister a Model from a JVM.

Dependencies

The profile option CZ: Add Model Routing Cookie must be set to True for Model routing to occur at runtime.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE unregister_model_from_pool (p_pool_identifier IN VARCHAR2,
                                      p_model_product_key IN VARCHAR2);
```

The following table describes the parameters for the UNREGISTER_MODEL_FROM_POOL function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the UNREGISTER_MODEL_FROM_POOL Function

Parameter	Data Type	Mode	Note
p_pool_identifier	varchar2	in	The JVM pool to which the specified Model was registered.
p_model_product_key	varchar2	in	Product Key of the Model that was registered. For details on Product Key, see Applicability Parameters, page 17-9.

UNREGISTER_POOL

This procedure unregisters a JVM pool, by deleting all the mappings in the mapping table (CZ_MODEL_POOL_MAPPINGS) that refer to the specified JVM pool. Uses an autonomous transaction.

Considerations Before Running

Use of this procedure assumes that you are routing Models to JVM pools, as described in Routing Models to Specified JVMs, page 20-10.

Timing

This procedure should be used when you need to unregister a JVM pool.

Dependencies

The profile option CZ: Add Model Routing Cookie must be set to True for Model routing to occur at runtime.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE unregister_pool (p_pool_identifier IN VARCHAR2);
```

The following table describes the parameters for the UNREGISTER_POOL function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the UNREGISTER_POOL Function

Parameter	Data Type	Mode	Note
p_pool_identifier	varchar2	in	The JVM pool to be unregistered.

UI_FOR_ITEM

This function returns a UI definition (`ui_def_id`) for a given inventory item (`inventory_item_id`) and organization item (`organization_id`) based on publication applicability parameters.

This function is used for backward compatibility. It calls CONFIG_UI_FOR_ITEM, page 17-23 with usage_name equal to "Any Usage" and publication_mode equal to 'P'.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a model to be returned. This function must be run on the instance that the model is published to.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION ui_for_item( inventory_item_id NUMBER,
                      organization_id NUMBER,
                      config_creation_date DATE,
                      ui_type VARCHAR2,
                      user_id NUMBER,
                      responsibility_id NUMBER,
                      calling_application_id NUMBER )
RETURN NUMBER;
```

The following table describes the parameters for the UI_FOR_ITEM function. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the UI_FOR_ITEM Function

Parameter	Data Type	Mode	Note
inventory_item_id	number	in	If the model was imported from Oracle BOM, then this is the Inventory Item ID for the published model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
organization_id	number	in	If the model was imported from Oracle BOM, then this is the organization ID for the published model, from the MTL_SYSTEM_ITEMS table, on which configuration models are based.
config_creation_date	date	in	This is the date the configuration was created.
ui_type	varchar2	in	This is the type of published UI sought and found for each product. Values are 'APPLET', 'DHTML', or 'JRAD'. If either DHTML or JRAD is passed, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If APPLET is passed, then the publication UI type can be either APPLET, DHTML, or JRAD. If DHTML or JRAD is passed and there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.

Parameter	Data Type	Mode	Note
user_id	number	in	This is the ID for the Oracle Applications user that is logged into from FND_USER.
responsibility_id	number	in	This is the responsibility that the Oracle Applications user had in the host application.
calling_application_id	number	in	The registered ID of an application for which the model is published. See Applicability Parameters, page 17-9.

Considerations After Running

None

Results

This function returns the user interface ID associated with the selected publication.

If the `ui_type` is APPLET, then the publication UI type can be either APPLET, DHTML, or JRAD.

If the `ui_type` is either DHTML or JRAD, then the publication UI type must be either DHTML or JRAD. Otherwise `NULL` is returned. If there is no publication available for the item, then the API returns the user interface ID of the BOM JRAD UI.

UI_FOR_PUBLICATION_ID

This function returns a UI definition (`ui_def_id`) for a specified publication ID.

Considerations Before Running

None

Timing

This function should be used after publishing Models, to verify if publication lookup will succeed for a given set of applicability parameters.

Dependencies

Publications must exist for a model to be returned. This function must be run on the

instance that the model is published to.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION ui_for_publication_id ( publication_id NUMBER )
RETURN NUMBER;
```

The following table describes the parameters for the UI_FOR_PUBLICATION_ID function. See Using the UI_FOR_PUBLICATION_ID Function, page 17-69 for an example of how these parameters are used. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the UI_FOR_PUBLICATION_ID Function

Parameter	Data Type	Mode	Note
publication_id	number	in	This is the specified publication id in the CZ_MODEL_PUBLICATIONS table.

Example

When called in SQL*Plus, this example prints out the ID of the UI definition associated with the publication identified by the `publication_id` parameter. If the publication has no associated UI, then a message is printed.

Using the UI_FOR_PUBLICATION_ID Function
Example

```
set serveroutput on
DECLARE
v_ui_def_id number;
BEGIN
-- The publication must have status of 'OK' ("Complete").
   v_ui_def_id := cz_cf_api.ui_for_publication_id(12345);
   IF v_ui_def_id IS NULL THEN
      dbms_output.put_line('UI Def ID: '||'NOT FOUND');
   ELSE
      dbms_output.put_line('UI Def ID: '||v_ui_def_id);
   END IF;
END;
```

VALIDATE

This procedure validates a configuration. You can use this procedure to check whether a configuration is still valid after an event that may cause it to become invalid. Such events might include the following:

- A change in the configuration rules
- The importing of the configuration from another system
- A change to the configuration inputs by another program
- The ordered configured BOM Items (input_list) do not match the batch validation BOM Items (from a previously processed configuration)

This procedure is a single call validation procedure that uses tables to exchange multi-valued data. A validation_status, page 17-71 and a table of XML messages are returned.

Considerations Before Running

None

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE VALIDATE ( config_input_list IN CFG_INPUT_LIST,
                     init_message IN VARCHAR2,
                     config_messages IN OUT NOCOPY CFG_OUTPUT_PIECES,
                     validation_status IN OUT NOCOPY NUMBER,
                     URL IN VARCHAR2 DEFAULT
FND_PROFILE.Value('CZ_UIMGR_URL'),
                     p_validation_type IN VARCHAR2 DEFAULT
CZ_API_PUB.VALIDATE_ORDER));
```

The following table describes the parameters for the VALIDATE procedure. This includes the data type, the mode (in or out), and a brief note about the parameter.

Parameters for the VALIDATE Procedure

Parameter	Data Type	Mode	Note
config_input_list	CFG_INPUT_LIST, page 17-12. See Custom Data Types, page 17-11 for a definition of this type.	in	This is a list of input selections.
init_message	varchar2	in	Initialization message

Parameter	Data Type	Mode	Note
config_messages	CFG_OUTPUT_PIECES, page 17-12 . See Custom Data Types, page 17-11 for a definition of this type.	out	This is a table of the output XML messages produced by validating the configuration.
validation_status	varchar2	out	The status code returned by validating the configuration: 0 - CONFIG_PROCESSED, page 17-72 1 - CONFIG_PROCESSED_NO_TERMINATE, page 17-72 2 - INIT_TOO_LONG, page 17-72 3 - INVALID_OPTION_REQUEST, page 17-72 4 - CONFIG_EXCEPTION, page 17-72 5 - DATABASE_ERROR, page 17-72 6 - UTL_HTTP_INIT_FAILED, page 17-72 7 - UTL_HTTP_REQUEST_FAILED, page 17-72
url	varchar2	in	The URL for the Oracle Configurator Servlet. Default will interrogate the current profile for this URL, using `FND_PROFILE.Value('CZ_UIMGR_URL')`.
p_validation_type	varchar2	in	The possible values are CZ_API_PUB.VALIDATE_ORDER, CZ_API_PUB.VALIDATE_FULFILLMENT, and CZ_API_PUB.INTERACTIVE. The default is CZ_API_PUB.VALIDATE_ORDER.

Example

For an example of how these parameters are used, see Calling the CZ_CF_API.VALIDATE Procedure, page 11-4.

Considerations After Running

None

Results

This procedure returns the values listed in the tableValues Returned by the VALIDATE Procedure., page 17-72

Values Returned by the VALIDATE Procedure

Return Value	Description
CONFIG_PROCESSED	Configuration processed successfully, and a termination message was returned.
CONFIG_PROCESSED_NO_TERMINATE	Configuration processed, but no termination message was returned.
INIT_TOO_LONG	Initialization message must be less than 2048 characters.
INVALID_OPTION_REQUEST	Returned when an input does not include a component code or quantity.
CONFIG_EXCEPTION	Unknown error
DATABASE_ERROR	Unknown error
UTL_HTTP_INIT_FAILED UTL_HTTP_REQUEST_FAILED	Procedure uses UTL_HTTP package to pass data to Configurator Servlet. These exceptions can be returned by UTL_HTTP procedures. See the documentation resources on *supplied PL/SQL packages* for additional information.

CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION

This procedure verifies that the specified configuration exists and returns whether it is valid or complete. This procedure functions like a view. The procedure queries the configuration data checking that the configuration exists in the CZ schema. This query provides essential information to downstream applications without directly querying the database.

Considerations Before Running

None

Timing

This procedure validates that the configuration header is a session header and not an instance header.

Dependencies

None

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE verify_configuration( p_api_version        IN  NUMBER,
                                p_config_hdr_id      IN  NUMBER,
                                p_config_rev_nbr     IN  NUMBER,
                                x_exists_flag        OUT NOCOPY
VARCHAR2,
                                x_valid_flag         OUT NOCOPY
VARCHAR2,
                                x_complete_flag      OUT NOCOPY
VARCHAR2,
                                x_return_status      OUT NOCOPY
VARCHAR2,
                                x_msg_count          OUT NOCOPY  NUMBER,
                                x_msg_data           OUT NOCOPY
VARCHAR2);
```

The following table describes the parameters for the CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION, page 17-72 procedure.

Parameters for the CZ_CONFIG_API_PUB.VERIFY_CONFIGURATION Procedure

Parameter	Data Type	Mode	Note
p_api_version	number	in	Required. See API Version Numbers, page 18-9.
p_config_hdr_id	number	in	Required. Header ID of the configuration to be verified.
p_config_rev_nbr	number	in	Required. Revision number of the configuration to be verified.
x_exists_flag	varchar2	out	If config_hdr_id and config_rev_nbr describe a saved configuration, then FND_API.G_TRUE is returned. If there is no saved configuration, then FND_API.G_FALSE is returned.

Parameter	Data Type	Mode	Note
x_valid_flag	varchar2	out	If the configuration exists and is valid, then FND_API.G_TRUE is returned. If the configuration exists but is invalid, then FND_API.G_FALSE is returned. If the configuration does not exist then NULL.
x_complete_flag	varchar2	out	If the configuration exists and is complete, then FND_API.G_TRUE is returned. If the configuration exists but is incomplete, then FND_API.G_FALSE is returned. If the configuration does not exist, then NULL.
x_return_status	varchar2	out	Must return FND_API.G_RET_STS_SUCCESS if procedure completed successfully; otherwise return FND_API.G_RET_STS_ERROR or FND_API.G_RET_STS_UNEXP_ERROR if an error occurs within the procedure
x_msg_count	number	out	The number of error messages returned in the `x_msg_data` parameter.
x_msg_data	varchar2	out	See corresponding parameter in Parameters for the CZ_CONFIG_API_PUB.COPY_CONFIGURATION Procedure, page 17-41.

18

Programmatic Tools for Maintenance

This chapter describes a set of programmatic tools (PL/SQL procedures) that you can use primarily to maintain a deployed runtime Oracle Configurator.

This chapter covers the following topics:

- Overview
- Overview of the CZ_modelOperations_pub Package
- Choosing the Right Tool for the Job
- Queries to Support the CZ_modelOperations_pub Package
- Reference for the CZ_modelOperations_pub Package
- CREATE_RP_FOLDER
- CREATE_UI
- CREATE_JRAD_UI
- DEEP_MODEL_COPY
- EXECUTE_POPULATOR
- FORCE_UNLOCK_MODEL
- FORCE_UNLOCK_TEMPLATE
- GENERATE_LOGIC
- IMPORT_SINGLE_BILL
- IMPORT_GENERIC
- PUBLISH_MODEL
- MIGRATE_MODELS
- REFRESH_SINGLE_MODEL
- REFRESH_UI
- REFRESH_JRAD_UI

- REPOPULATE
- REPUBLISH_MODEL
- RP_FOLDER_EXISTS

Overview

This chapter describes a set of programmatic tools that you can use primarily to maintain a deployed runtime Oracle Configurator. This includes:

- Choosing the Right Tool for the Job, page 18-3
- Queries to Support the CZ_modelOperations_pub Package, page 18-5
- Reference for the CZ_modelOperations_pub Package, page 18-9

> **Important:** For the latest reference information on these APIs, see the Oracle Integration Repository, which is installed with your patched instance of the Oracle E-Business Suite, as described in the Preface of this guide. In the Integration Repository, the package described in this chapter can be located by using the Search function on the Internal Name `CZ_MODELOPERATIONS_PUB`.

> **Important:** This chapter includes references to DHTML user interfaces, but these are temporarily retained for historical informational purposes only. As of this release, DHTML UIs are no longer supported.

For information on tools for developing a configuration model or deploying a runtime Oracle Configurator, see Programmatic Tools for Development, page 17-1.

Overview of the CZ_modelOperations_pub Package

The programmatic tools that you use to maintain a deployed runtime Oracle Configurator are provided in the PL/SQL package CZ_modelOperations_pub.

Purpose of the Package

The CZ_modelOperations_pub package contains a set of APIs that enable you to automate day-to-day maintenance activities, thus reducing the maintenance workload. The operations covered by this are:

- Importing and refreshing configuration models with data from Oracle Applications BOMs

- Migrating Models to another development instance
- Generation and refreshing of logic and User Interfaces
- Publication of generated logic and User Interfaces
- Initial execution and refreshing of Item Master Populators
- Force unlocking of Models in Oracle Configurator
- Force unlocking of User Interface Content Templates in Oracle Configurator

Installation of the Package

The information provided for the package CZ_CF_API in Installation of the Packages, page 17-5 also applies to the package CZ_modelOperations_pub.

References for Working with PL/SQL Procedures and Functions

For background information and details on basic aspects of working with the PL/SQL procedures and functions in this package, see References for Working with PL/SQL Procedures and Functions, page 17-6 in References for Working with PL/SQL Procedures and Functions, page 17-6, which suggests relevant topics in the Oracle Documentation Library.

Choosing the Right Tool for the Job

Use the table below to choose the appropriate procedure or function for the task you want to perform. These procedures and functions are described in detail in Procedures and Functions in the CZ_modelOperations_pub Package, page 18-11.

Uses of Procedures and Functions in the CZ_modelOperations_pub package

Area	For This Purpose ...	Use This Procedure or Function ...
Repository	To create a folder in the Repository, or check whether a folder exists	CREATE_RP_FOLDER, page 18-12 RP_FOLDER_EXISTS, page 18-38

Area	For This Purpose ...	Use This Procedure or Function ...
Models	To import, refresh, or migrate Models	IMPORT_SINGLE_BILL, page 18-26
		IMPORT_GENERIC, page 18-27
		MIGRATE_MODELS, page 18-30
		REFRESH_SINGLE_MODEL, page 18-32
	To make a deep copy of a specified Model	DEEP_MODEL_COPY, page 18-19
	To publish or republish Models	PUBLISH_MODEL, page 18-29
		REPUBLISH_MODEL, page 18-37
	To run Populators	EXECUTE_POPULATOR, page 18-20
		REPOPULATE, page 18-35
	To force unlock a Model	FORCE_UNLOCK_MODEL, page 18-21
Rules	To generate logic	GENERATE_LOGIC, page 18-25
User Interfaces	To generate or refresh a user interface	CREATE_JRAD_UI, page 18-17
		REFRESH_JRAD_UI, page 18-34
		CREATE_UI, page 18-14 (DHTML or Java **Applet** UI)
		REFRESH_UI, page 18-33 (DHTML or Java Applet UI)
	To force unlock a UI Content Template	FORCE_UNLOCK_TEMPLATE, page 18-23

Queries to Support the CZ_modelOperations_pub Package

This section contains PL/SQL queries that indicate the values you need to provide as parameters to certain procedures in the CZ_modelOperations_pub package.

Querying for Model and Folder IDs

You can determine the IDs of Models and folders in the Repository of Oracle Configurator Developer by customizing a View so that it displays the column **DatabaseId**. See the *Oracle Configurator Developer User's Guide* for details on customizing Views.

You can also use a database query to list these IDs. Query for Models and Folders, page 18-6 provides a SQL query that lists the names and IDs of source (not published) Models, and the folders that contain them in the Repository of Oracle Configurator Developer.

The ID of a Model is stored as CZ_DEVL_PROJECTS.DEVL_PROJECT_ID. This query selects a value for DEVL_PROJECT_ID. This ID can then be used as a value for the parameter `p_devl_project_id or p_model_id` to the following procedures:

- CREATE_JRAD_UI, page 18-17
- CREATE_UI, page 18-14
- DEEP_MODEL_COPY, page 18-19
- FORCE_UNLOCK_MODEL, page 18-21
- GENERATE_LOGIC, page 18-25
- REFRESH_SINGLE_MODEL, page 18-32
- REPOPULATE, page 18-35

The ID of a folder that contains a specified Model is stored as CZ_RP_ENTRIES.ENCLOSING_FOLDER. This query selects a value for ENCLOSING_FOLDER. This ID can then be used as a value for the parameter `p_encl_folder_id` to the following procedures:

- CREATE_RP_FOLDER, page 18-12
- RP_FOLDER_EXISTS, page 18-38

Query for Models and Folders
Example

```
select
  P.devl_project_id,
  P.name,
  R.enclosing_folder,
  R2.name FOLDER
from
  cz_devl_projects P,
  cz_rp_entries R,
  cz_rp_entries R2
where
  R.object_type = 'PRJ' and
  R.deleted_flag = '0' and
  P.deleted_flag = '0' and
  P.devl_project_id = R.object_id and
  R2.object_id = R.enclosing_folder and
  R2.object_type ='FLD';
```

You can add the following condition to the beginning of the WHERE clause of this query to specify the name of a particular Model as it appears in Oracle Configurator Developer.

Example

```
P.name like '%your Model's name%' and
```

You can add the following condition to the beginning of the WHERE clause of this query to specify the name of a particular folder as it appears in Oracle Configurator Developer.

Example

```
R2.name like 'your folder's name%' and
```

Querying for User Interface IDs

You can determine the IDs of User Interfaces by examining the **UI ID** column in the User Interfaces area of the Workbench of Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for details on customizing Views.

You can also use a database query to list these IDs. Query for User Interface IDs, page 18-7 provides a SQL query that lists the names and IDs of available user interfaces for a specified Model. To determine the `devl_project_ID` for the specified Model, use the query in Query for Models and Folders, page 18-6.

This query selects values for the column CZ_UI_DEFS.UI_DEF_ID. This UI_DEF_ID is returned by the procedures CREATE_UI, page 18-14 and CREATE_JRAD_UI, page 18-17. You would use this ID as a value for the `p_ui_def_id` parameter for the procedures REFRESH_UI, page 18-33 and REFRESH_JRAD_UI, page 18-34.

Query for User Interface IDs Example

```
select
  ui_def_id,
  name
from
  cz_ui_defs
where
  devl_project_id = devl_project_ID
and
  deleted_flag = '0';
```

Querying for Referenced User Interface IDs

Query for Referenced DHTML and Java Applet User Interface IDs, page 18-7 provides a SQL query that lists the UIs for a given Model and all referenced Models of the given Model.

Query for Referenced DHTML and Java Applet User Interface IDs, page 18-7 provides a SQL query that lists the IDs of available referenced (child)DHTML and Java Applet user interfaces for a specified *parent_ui_def_ID*. To determine the *parent_ui_def_ID* for a specified Model, use the query in Query for User Interface IDs, page 18-7.

This query selects a value for the column CZ_UI_NODES.UI_DEF_ID. Use this value as a parameter for the following procedures:

- REFRESH_UI, page 18-33

Query for Referenced DHTML and Java Applet User Interface IDs Example

```
select distinct
  ui_def_id
from
  cz_ui_nodes
where
  cz_ui_nodes.deleted_flag = '0'
start with
  ui_def_id = parent_ui_def_ID
connect by
  prior cz_ui_nodes.ui_def_ref_id = cz_ui_nodes.ui_def_id
  and prior deleted_flag = '0'
order by
  cz_ui_nodes.ui_def_id;
```

Querying for Populators

Query for Populators, page 18-8 provides a SQL query that lists the names and IDs of Populators for a given Model.

To determine the *devl_project_ID_for_model* for the specified Model, use the query in Query for Models and Folders, page 18-6.

This query selects a value for the column CZ_POPULATORS.POPULATOR_ID. Use this value as a parameter for the following procedures:

- EXECUTE_POPULATOR, page 18-20

Query for Populators
Example

```
select
  populator_id,
  a.name POPULATOR_NAME,
  b.ps_node_id,
  b.name
from
  cz_populators a,
  cz_ps_nodes b
where
  a.owned_by_node_id = b.ps_node_id
and
  b.devl_project_id = devl_project_ID_for_model
and
  a.deleted_flag = '0'
  and b.deleted_flag = '0';
```

Querying for Error and Warning Information

Query for Error and Warning Information, page 18-9 provides a SQL query that retrieves the error and warning information that is recorded in the table CZ_DB_LOGS after you run one of the following procedures:

- CREATE_UI, page 18-14
- CREATE_JRAD_UI, page 18-17
- CREATE_RP_FOLDER, page 18-12
- DEEP_MODEL_COPY, page 18-19
- EXECUTE_POPULATOR, page 18-20
- FORCE_UNLOCK_MODEL, page 18-21
- FORCE_UNLOCK_TEMPLATE, page 18-23
- GENERATE_LOGIC, page 18-25
- IMPORT_GENERIC, page 18-27
- IMPORT_SINGLE_BILL, page 18-26
- MIGRATE_MODELS, page 18-30
- PUBLISH_MODEL, page 18-29
- REFRESH_JRAD_UI, page 18-34

- REFRESH_SINGLE_MODEL, page 18-32
- REFRESH_UI, page 18-33
- REPOPULATE, page 18-35
- REPUBLISH_MODEL, page 18-37

This query selects values for the columns URGENCY, STATUSCODE, and MESSAGE from the table CZ_DB_LOGS.

URGENCY and STATUSCODE only have significant values when populated by the GENERATE_LOGIC, page 18-25 procedure. The URGENCY values used by are 0 for errors and 1 for warnings. STATUSCODE values are not meaningful to the user but are important to the Oracle Configurator engineering team for the debugging of logic generation code.

Query for Error and Warning Information Example

```
select
  urgency,
  statuscode,
  message
from
  cz_db_logs
where
  run_id = run_ID_returned_from_procedure;
```

Reference for the CZ_modelOperations_pub Package

- This section provides descriptions of each of the procedures in the CZ_modelOperations_pub package. These procedures are listed alphabetically in Procedures and Functions in the Package CZ_modelOperations_pub, page 18-11.
- Descriptions of the custom data types defined in the package are also provided, in Custom Data Types, page 18-9.
- For a basic example of how to call one of the functions in the CZ_CF_API package, see Using the GENERATE_LOGIC Procedure, page 18-26.
- See also Overview of the CZ_modelOperations_pub Package, page 18-2.

Custom Data Types

There are no custom data types defined in the CZ_modelOperations_pub package.

API Version Numbers

Oracle APIs incorporate a mechanism called **API** version numbers. This mechanism:

- Allows an API to differentiate between changes that require you to change your API calling code and those that don't.
- Allows an API to detect incompatible calls.
- Allows you to quickly determine if calling a new version of an API requires you to change any of your code.
- Allows you to easily figure out which version of an API you need to call to take advantage of new features.

Format of API Version Numbers

API version numbers consist of two segments separated by a decimal point. The first segment is the major version number; the second segment is the minor version number. The starting version number for an API is always 1.0.

The following table shows an example of an API Version number and the major and minor version derived from the API version.

API Version Number	Major Version	Minor Version
1.0	1	0
2.4	2	4

If the major version number has changed, then you probably need to modify your programs that call that API. Major version changes include changes to the list of required parameters or changing the value of an API OUT parameter.

If only the minor version number has changed, then you probably do not need to modify your programs.

Current API Version Number for This Package

The API version number for the APIs included in the current version of the CZ_modelOperations_pub package is:

```
1.0
```

The local constant that stores this version number is:

```
l_api_version   CONSTANT NUMBER
```

Checking for Incompatible API Calls

To detect incompatible calls, programs calling an API must pass an API version number as one of the input parameters. The API can then compare the passed version number to its current version number, and detect any incompatible calls.

The Oracle standard parameter used by all procedures in this package to pass in the API version number is:

Example

```
p_api_version IN NUMBER
```

This parameter is required, and has no initial values, thus forcing your program to pass this parameter when calling an API.

If your call to the API results in a version incompatibility, then an error message is inserted in the table CZ_DB_LOGS. You can examine the message using a query like the one shown in Query for Error and Warning Information, page 18-9.

Procedures and Functions in the CZ_modelOperations_pub Package

This section provides descriptions of each of the procedures and functions in the CZ_modelOperations_pub package, arranged alphabetically. These procedures and functions are listed in Procedures and Functions in the Package CZ_modelOperations_pub, page 18-11.

The following table lists the API procedures and functions in the CZ_modelOperations_pub package.

Procedures and Functions in the Package CZ_modelOperations_pub

API Name	P/FP = procedure, F = function
CREATE_RP_FOLDER, page 18-12	P
CREATE_UI, page 18-14	P
CREATE_JRAD_UI, page 18-17	P
DEEP_MODEL_COPY, page 18-19	P
EXECUTE_POPULATOR, page 18-20	P
FORCE_UNLOCK_MODEL, page 18-21	P
FORCE_UNLOCK_TEMPLATE, page 18-23	P
GENERATE_LOGIC, page 18-25	P

API Name	P/FP = procedure, F = function
IMPORT_SINGLE_BILL, page 18-26	P
IMPORT_GENERIC, page 18-27	P
MIGRATE_MODELS, page 18-30	P
PUBLISH_MODEL, page 18-29	P
REFRESH_SINGLE_MODEL, page 18-32	P
REFRESH_UI, page 18-33	P
REFRESH_JRAD_UI, page 18-34	P
REPOPULATE, page 18-35	P
REPUBLISH_MODEL, page 18-37	P
RP_FOLDER_EXISTS, page 18-38	F

CREATE_RP_FOLDER

The CREATE_RP_FOLDER procedure creates a new folder in the specified enclosing (parent) folder of the Repository of Oracle Configurator Developer.

If a folder with the same name already exists in the enclosing folder, then that folder's ID is returned in the `x_new_folder_id` parameter. You can use the function RP_FOLDER_EXISTS, page 18-38 to determine beforehand whether a folder exists.

See also:

- RP_FOLDER_EXISTS, page 18-38

Considerations Before Running

None

Alternatives

As an alternative to using this procedure, you can create a folder in Oracle Configurator Developer, by using the **Create** icon in the Repository. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE create_rp_folder(p_api_version          IN  NUMBER
                          ,p_encl_folder_id       IN
CZ_RP_ENTRIES.OBJECT_ID%TYPE
                          ,p_new_folder_name      IN
CZ_RP_ENTRIES.NAME%TYPE
                          ,p_folder_desc          IN

CZ_RP_ENTRIES.DESCRIPTION%TYPE
                          ,p_folder_notes         IN
CZ_RP_ENTRIES.NOTES%TYPE
                          ,x_new_folder_id        OUT NOCOPY
CZ_RP_ENTRIES.OBJECT_ID%TYPE
                          ,x_return_status        OUT NOCOPY  VARCHAR2
                          ,x_msg_count            OUT NOCOPY  NUMBER
                          ,             OUT NOCOPY  VARCHAR2
                          );
```

The following table describes the parameters for the CREATE_RP_FOLDER procedure. This includes the mode (in or out), the data type, and a brief note about the parameter.

Parameters for the CREATE_RP_FOLDER Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9 .
p_encl_folder_id	in	number	Required. The ID of the enclosing (parent) folder in which you are creating the new folder. To determine the ID of a folder, see Querying for Model and Folder IDs, page 18-5. To specify the root folder of the Repository, use the constant RP_ROOT_FOLDER.
p_new_folder_name	in	varchar2	Required. The name of the new folder that you are creating.

Parameter	Mode	Data Type	Note
p_folder_desc	in	varchar2	A description for the new folder that you are creating
p_folder_notes	in	varchar2	Notes text for the new folder that you are creating
x_new_folder_id	out	number	The ID of the new folder created. If a folder with the same new name already exists in the enclosing folder, the ID of that existing folder.
x_return_status	out	varchar2	Either FND_API.G_RET_STS_ERROR, FND_API.G_RET_STS_SUCCESS, FND_API.G_RET_STS_UNEXP_ERROR.
x_msg_count	out	number	The number of error messages returned in the `x_msg_data` parameter.
x_msg_data	out	varchar2	A string that contains any error messages.

CREATE_UI

The CREATE_UI procedure generates a new user interface for a model. This procedure generates only legacy Configurator User Interfaces (DHTML or Java applet) of the type generated with the limited edition of Oracle Configurator Developer.

If referenced models are present, then the behavior is the following:

1. If a referenced model has one or more user interfaces of the input UI style (DHTML or Applet), then the root UI will refer to the last UI created with this style.
2. If a referenced model has no user interface, the procedure will generate a new UI for that model.

See also:

- REFRESH_UI, page 18-33
- CREATE_JRAD_UI, page 18-17

Considerations Before Running

None

Alternatives

As an alternative to using this procedure, you can create a UI in the limited edition of Oracle Configurator Developer. For more information see the Oracle Configurator Release Notes for this release.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE create_ui(p_api_version IN NUMBER,
                    p_devl_project_id  IN  NUMBER,
                    x_ui_def_id OUT NOCOPY NUMBER,
                    x_run_id OUT NOCOPY NUMBER,
                    x_status OUT NOCOPY NUMBER,
                    p_ui_style IN VARCHAR2 DEFAULT 'COMPONENTS',
                    p_frame_allocation IN NUMBER DEFAULT 30,
                    p_width IN NUMBER DEFAULT 640,
                    p_height IN NUMBER DEFAULT 480,
                    p_show_all_nodes   IN  VARCHAR2 DEFAULT '0',
                    p_look_and_feel    IN  VARCHAR2 DEFAULT 'BLAF',
                    p_wizard_style     IN  VARCHAR2 DEFAULT '0',
                    p_max_bom_per_page IN  NUMBER   DEFAULT 10,
                    p_use_labels       IN  VARCHAR2 DEFAULT '1');
```

The following table describes the parameters for the CREATE_UI procedure. This includes the mode (in or out), the data type, and a brief note about the parameter.

Parameters for the CREATE_UI Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9 .
p_devl_project_id	in	number	The ID of the Model for which to create a UI. See Query for Models and Folders, page 18-6 for a query that provides this ID (DEVL_PROJECT_ID).
x_ui_def_id	out	number	The ID of the UI that is created. This is stored as CZ_UI_DEFS.UI_DEF_ID.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored.

Parameter	Mode	Data Type	Note
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.
p_ui_style	in	varchar2	The style of the UI. Values are: '0' or 'COMPONENTS' for a Component Tree (DHTML) style, '3' or 'APPLET' for an Applet UI style. The default is 'COMPONENTS'.
p_frame_allocation	in	number	The left-hand frame allocation for the new UI, in %. The default is 30 (30% of the screen allocated to the left-hand frame).
p_width	in	number	The width of the screens in the new UI, in pixels. The default is 640.
p_height	in	number	The height of the screens in the new UI, in pixels. The default is 480.
p_show_all_nodes	in	varchar2	Controls whether the "include in generated UI" flag on Model nodes is respected. If this parameter is '1', then the new UI will include all Model nodes including those marked as "do not include in generated UI". If this parameter is '0', then the new UI will respect the "include in generated UI" flag on Model nodes. The default is '0'.
p_look_and_feel	in	varchar2	The look and feel for the new UI. Values are: 'BLAF', 'APPLET', or 'FORMS'. The default is 'BLAF'. 'FORMS' can only be used if p_ui_style, page 18-16 is 'COMPONENTS'. The default is 'BLAF'.
p_wizard_style	in	varchar2	Whether to generate wizard style navigation. Values are: '0' for No, '1' for Yes. The default is '0' (No).
p_max_bom_per_page	in	number	The maximum number of BOM Option Class children per screen. The default is 10.

Parameter	Mode	Data Type	Note
p_use_labels	in	varchar2	Indicates how to generate captions: '0' for description only, '1' for name only, '2', for name and description. The default is '1'.

CREATE_JRAD_UI

The CREATE_JRAD_UI procedure generates a new User Interface for a Model. This procedure generates only User Interfaces that are based on the OA Framework. For more information on the OA Framework, see the Oracle Application Framework Documentation Resources, Release 12, on the Oracle Support Web site.

See also:

- REFRESH_JRAD_UI, page 18-34
- CREATE_UI, page 18-14

Considerations Before Running

None

Alternatives

As an alternative to using this procedure, you can create a UI in Oracle Configurator Developer, in the UI area of the Workbench. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE create_jrad_ui(p_api_version        IN  NUMBER,
                         p_devl_project_id    IN  NUMBER,
                         p_show_all_nodes     IN  VARCHAR2,
                         p_master_template_id IN  NUMBER,
                         p_create_empty_ui    IN  VARCHAR2,
                         x_ui_def_id          OUT NOCOPY NUMBER,
                         x_return_status      OUT NOCOPY VARCHAR2,
                         x_msg_count          OUT NOCOPY NUMBER,
                                    OUT NOCOPY VARCHAR2);
```

The following table describes the parameters for the CREATE_JRAD_UI procedure. This includes the mode (in or out), the data type, and a brief note about the parameter.

Parameters for the CREATE_JRAD_UI Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9 .
p_devl_project_id	in	number	The ID of the Model for which to create a UI. See Query for Models and Folders, page 18-6 for a query that provides this ID (DEVL_PROJECT_ID).
p_show_all_nodes	in	varchar2	'Controls whether the "include in generated UI" flag on Model nodes is respected. If this parameter is '1', then the new UI will include all Model nodes including those marked as "do not include in generated UI". If this parameter is '0', then the new UI will respect the "include in generated UI" flag on Model nodes.The default is '0'.
p_master_template_id	in	number	You can determine the IDs of UI master Templates in the Repository of Oracle Configurator Developer by customizing a View so that it displays the column **DatabaseId**. See the *Oracle Configurator Developer User's Guide* for details on customizing Views.
p_create_empty_ui	in	varchar2	If this parameter is '1', then the new UI will be an "empty" UI. See the *Oracle Configurator Developer User's Guide* for details on empty UIs.
x_ui_def_id	out	number	The ID of the UI that is created. This is stored as CZ_UI_DEFS.UI_DEF_ID.
x_return_status	out	varchar2	Either FND_API.G_RET_STS_ERROR, FND_API.G_RET_STS_SUCCESS, FND_API.G_RET_STS_UNEXP_ERROR
x_msg_count	out	number	The number of error messages returned in the `x_msg_data` parameter.
x_msg_data	out	varchar2	A string that contains any error messages.

DEEP_MODEL_COPY

The DEEP_MODEL_COPY procedure performs a deep copy of a specified Model.

Deep copying creates a new copy of the specified Model, along with new copies of any referenced Models. You can choose to copy the Model without its configuration rules, user interfaces, or referenced child Models.

Considerations Before Running

None

Alternatives

As an alternative to using this procedure, you can perform a deep copy of a Model in Oracle Configurator Developer, by using the **Copy** command in the Repository. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE deep_model_copy(p_api_version IN  NUMBER,
                          p_devl_project_id    IN  NUMBER,
                          p_folder      IN  NUMBER,
                          p_copy_rules  IN  NUMBER,
                          p_copy_uis    IN  NUMBER,
                          p_copy_root   IN  NUMBER,
                          x_devl_project_id    OUT NOCOPY NUMBER,
                          x_run_id      OUT NOCOPY NUMBER,
                          x_status      OUT NOCOPY NUMBER);
```

The table Parameters for the DEEP_MODEL_COPY Procedure, page 18-19 describes the parameters for the DEEP_MODEL_COPY procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the DEEP_MODEL_COPY Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_devl_project_id	in	number	The ID of the Model of which a copy is to be made. See Query for Models and Folders, page 18-6 for a query that provides this ID (DEVL_PROJECT_ID).

Parameter	Mode	Data Type	Note
p_folder	in	number	The folder to which the copy is made. See Query for Models and Folders, page 18-6 for a query that provides this number (ENCLOSING_FOLDER).
p_copy_rules	in	number	Set to 1 to copy configuration rules with the model, 0 to omit the rules.
p_copy_uis	in	number	Set to 1 to copy user interfaces with the model, 0 to omit the user interfaces.
p_copy_root	in	number	Set to 1 to copy only the root model, 0 to copy all referenced models.
x_devl_project_id	out	number	The ID (DEVL_PROJECT_ID) of the Model created by the copying operation.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

EXECUTE_POPULATOR

The EXECUTE_POPULATOR procedure can be used to refresh the CZ_PS_NODES table by implementing a Populator.

A Populator is a mechanism that automatically builds Model structure from data in the Item Master. See the *Oracle Configurator Developer User's Guide* for more details on Populators.

The CZ_PS_NODES table in the CZ schema describes the structure of the generated logic.

See the description of REPOPULATE, page 18-35 for information on the related procedure for repopulating Model structure.

Considerations Before Running

Before running the EXECUTE_POPULATOR procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, you can define and run a Populator using Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for instructions on using Populators.

Another alternative to using this procedure is to run the Execute Populators in Model concurrent program. See Execute Populators in Model Concurrent Program, page C-29 for details on running this concurrent program.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE execute_populator(p_api_version IN NUMBER,
                            p_populator_id IN NUMBER,
                            p_imp_run_id IN OUT NOCOPY VARCHAR2,
                            x_run_id OUT NOCOPY NUMBER,
                            x_status OUT NOCOPY NUMBER);
```

The following table describes the parameters for the EXECUTE_POPULATOR procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the EXECUTE_POPULATOR Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_populator_id	in	number	The value of CZ_POPULATORS.POPULATOR_ID for the Populator to be used.
p_imp_run_id	in/out	varchar2	Stored in CZ_IMP_PS_NODES.RUN_ID.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

FORCE_UNLOCK_MODEL

The FORCE_UNLOCK_MODEL procedure unlocks one or more Models according to

user-defined criteria.

Model locking provides a mechanism that protects multiple users from modifying the same Model at the same time. The FORCE_UNLOCK_MODEL procedure only works when it is run as the user who has access to the force unlock functionality. See the *Oracle Configurator Developer User's Guide* for more information on Model locking.

Considerations Before Running

Before running the FORCE_UNLOCK_MODEL procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, the Oracle Configurator Administrator can unlock any object that is locked by another user. See the *Oracle Configurator Developer User's Guide* for more information on force unlocking.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE force_unlock_model(p_api_version IN NUMBER,
                             p_model_id IN NUMBER,
                             p_unlock_references IN VARCHAR2,
                             p_init_msg_list IN VARCHAR2,
                             x_return_status OUT NOCOPY VARCHAR2,
                             x_msg_count OUT NOCOPY NUMBER,
                             x_msg_data OUT NOCOPY VARCHAR2);
```

The following table describes the parameters for the FORCE_UNLOCK_MODEL procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the FORCE_UNLOCK_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_model_id	in	number	Required. The value of CZ_DEVL_PROJECTS.MODEL_ID for the Model to be unlocked.

Parameter	Mode	Data Type	Note
p_unlock_references	in	varchar2	Controls whether to unlock just the Model or to unlock the Model and the entire tree of referenced Models. The values are FND_API.G_TRUE or FND_API.G_FALSE If this parameter is FND_API.G_FALSE, then the just the Model is unlocked. The default is FND_API.G_FALSE.
p_init_msg_list	in	varchar2	Either FND_API.G_TRUE if the FND stack should be initialized, or FND_API.G_FALSE if the FND stack should not be initialized.
x_return_status	out	varchar2	Either FND_API.G_RET_STS_ERROR, FND_API.G_RET_STS_SUCCESS, FND_API.G_RET_STS_UNEXP_ERROR.
x_msg_count	out	number	The number of error messages that are available on the FND error stack after the completion of the procedure.
x_msg_data	out	varchar2	A string that contains any error messages.

FORCE_UNLOCK_TEMPLATE

The FORCE_UNLOCK_TEMPLATE procedure unlocks a UI Content Template.

Locking UI Content Templates provides a mechanism that protects multiple users from modifying the same UI Content Template at the same time. The FORCE_UNLOCK_TEMPLATE API only works when it is run as the user who has access to the force unlock functionality. See the *Oracle Configurator Developer User's Guide* for more information on UI Content Template locking.

Considerations Before Running

Before running the FORCE_UNLOCK_TEMPLATE procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, the Oracle Configurator Administrator can unlock any object that is locked by another user. See the *Oracle Configurator Developer User's Guide* for more information on force unlocking.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE force_unlock_template(p_api_version IN NUMBER,
                           p_template_id IN NUMBER,
                           p_init_msg_list IN VARCHAR2,
                           x_return_status OUT NOCOPY VARCHAR2,
                           x_msg_count OUT NOCOPY NUMBER,
                           x_msg_data OUT NOCOPY VARCHAR2);
```

The following table describes the parameters for the FORCE_UNLOCK_TEMPLATE procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the FORCE_UNLOCK_TEMPLATE Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_template_id	in	number	Required. The value of CZ_UI_TEMPLATES.TEMPLATE_ID for the UI Content Template to be unlocked.
p_init_msg_list	in	varchar2	Either FND_API.G_TRUE if the FND stack should be initialized, or FND_API.G_FALSE if the FND stack should not be initialized.
x_return_status	out	varchar2	Either FND_API.G_RET_STS_ERROR, FND_API.G_RET_STS_SUCCESS, FND_API.G_RET_STS_UNEXP_ERROR.
x_msg_count	out	number	The number of error messages that are available on the FND error stack after the completion of the procedure.
x_msg_data	out	varchar2	A string that contains any error messages.

GENERATE_LOGIC

The GENERATE_LOGIC procedure generates the logic for a Model and all of its referenced Models if necessary.

Considerations Before Running

Before running the GENERATE_LOGIC procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, you can generate logic in Oracle Configurator Developer, in the General area of the Workbench. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE generate_logic(p_api_version IN NUMBER,
                         p_devl_project_id IN  NUMBER,
                         x_run_id OUT NOCOPY NUMBER,
                         x_status OUT NOCOPY NUMBER);
```

The following table describes the parameters for the GENERATE_LOGIC procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the GENERATE_LOGIC Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_devl_project_id	in	number	The ID of the Model for which to generate logic. See Query for Models and Folders, page 18-6 for a query that provides this ID (DEVL_PROJECT_ID).
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored.

Parameter	Mode	Data Type	Note
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_WARNING, or G_STATUS_SUCCESS.

Example

When called in SQL*Plus, this example generates logic for a model with the ID (DEVL_PROJECT_ID) specified by the `p_devl_project_id` parameter. After the procedure runs, it prints the run ID and status.

Using the GENERATE_LOGIC Procedure Example

```
set serveroutput on
declare
x_run_id number;
x_status varchar2(100);
begin
CZ_modelOperations_pub.generate_logic(1.0,12345,x_run_id,x_status);
dbms_output.put_line('Run id: '||x_run_id);
dbms_output.put_line('x_status: '||x_status);
end;
```

IMPORT_SINGLE_BILL

The IMPORT_SINGLE_BILL procedure can be used to import a model from Oracle Bills of Materials (BOM).

See also:

- IMPORT_GENERIC, page 18-27

Considerations Before Running

Before running the IMPORT_SINGLE_BILL procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, you can run the Populate Configuration Models concurrent program. See Populate and Refresh Configuration Models Concurrent Programs, page C-18 program for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE import_single_bill(p_api_version IN  NUMBER,
                             p_org_id IN NUMBER,
                             p_top_inv_item_id IN NUMBER,
                             x_run_id OUT NOCOPY NUMBER,
                             x_status OUT NOCOPY NUMBER);
```

The following table describes the parameters for the IMPORT_SINGLE_BILL procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the IMPORT_SINGLE_BILL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_org_id	in	number	Required. The organization ID of the bill to be imported.
p_top_inv_item_id	in	number	The Inventory Item ID of the top item to be imported (the BOM root).
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

IMPORT_GENERIC

The IMPORT_GENERIC procedure processes and imports data from the CZ interface tables as part of a custom import. See Custom Import, page 5-30 for details about custom (generic) import.

See also:

- IMPORT_SINGLE_BILL, page 18-26

Considerations Before Running

Before running the IMPORT_GENERIC procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, you can run the Populate Configuration Models concurrent program. See Populate and Refresh Configuration Models Concurrent Programs, page C-18 program for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE import_generic(p_api_version      IN  NUMBER
                        ,p_run_id          IN  NUMBER
                        ,p_rp_folder_id    IN  NUMBER
                        ,x_run_id          OUT NOCOPY NUMBER
                        ,x_status          OUT NOCOPY NUMBER);
```

The following table describes the parameters for the IMPORT_GENERIC procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the IMPORT_GENERIC Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9 .
p_run_id	in	number	Required. The Run ID generated by previously populating the import (CZ_IMP_*) tables. Specify the ID of the records that you want to process during a particular generic import session. If this ID is NULL, then all the records in the import tables where run_id is NULL will be processed. You should obtain the Run ID from the sequence CZ_XFR_RUN_INFOS_S, to avoid possible conflicts with the IMPORT_SINGLE_BILL, page 18-26 procedure.

Parameter	Mode	Data Type	Note
p_rp_folder_id	in	number	Required. The ID of the folder in the Repository into which you want to import the Model. To determine the ID of a folder, see Querying for Model and Folder IDs, page 18-5. To specify the root folder of the Repository, use the constant RP_ROOT_FOLDER.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored. Used to get results from CZ_XFR_RUN_INFOS and CZ_XFR_RUN_RESULTS.
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_SUCCESS, or G_STATUS_WARNING.

PUBLISH_MODEL

After a publication record is created in Oracle Configurator Developer, the PUBLISH_MODEL procedure exports the publication to the target database (that is, Model and UI data).

Considerations Before Running

Before running the PUBLISH_MODEL procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Restrictions and Limitations

This procedure should only be run on publications with a status of Pending.

Alternatives

As an alternative to using this procedure, you can publish models in Oracle Configurator Developer in the Publications area of the Repository. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE publish_model(p_api_version IN  NUMBER,
                        p_publication_id IN NUMBER,
                        x_run_id OUT NOCOPY NUMBER,
                        x_status OUT NOCOPY NUMBER);
```

The following table describes the parameters for the PUBLISH_MODEL procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the PUBLISH_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_publication_id	in	number	The publication ID generated when you publish a model in Oracle Configurator Developer , stored as CZ_MODEL_PUBLICATIONS.PUBLICATION_ID.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

MIGRATE_MODELS

The MIGRATE_MODELS procedure copies the model's structure, rules, UIs, UI Content Templates, UI Master Templates, Usages, Effectivity Sets, Configurator Extension Archives, Populators, corresponding Item Master, and Properties to another development instance.

Considerations Before Running

None

Restrictions and Limitations

None

Alternatives

As an alternative to using this procedure, you can migrate Models from Oracle Configurator Developer .. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE migrate_models(p_api_version IN  NUMBER,
                         p_model_list_tab  IN  cz_pb_mgr.t_ref,
                         p_tgt_instance_id IN
cz_servers.server_local_id%TYPE,
                         p_tgt_folder_id   IN
cz_rp_entries.object_id%TYPE,
p_enable_shallow  IN  VARCHAR2,
p_user_id         IN  NUMBER,
                           p_resp_id          IN  NUMBER,
                           p_appl_id          IN  NUMBER,
                           p_run_id           IN  NUMBER,
                           x_run_id           OUT NOCOPY NUMBER,
                           x_return_status    OUT NOCOPY VARCHAR2,
                           x_msg_count        OUT NOCOPY NUMBER,
                           x_msg_data         OUT NOCOPY VARCHAR2
                          )
;
```

The following table describes the parameters for the MIGRATE_MODELS procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the MIGRATE_MODELS Procedure

Parameter	Mode	Data Type	Note
p_model_list_tab	in	number	Table of numbers that contains a list of all Models selected for migration from the source instance..
p_tgt_instance_id	in	number	Identifies the database instance where the Models will be migrated.
p_tgt_folder_id	in	number	Identifies the remote Repository folder where the Models are migrated. This is the object_id in the target's cz_rp_entries table.
p_userid	in	number	Standard parameters required for locking.
p_respid			
p_applid			

Parameter	Mode	Data Type	Note
p_run_id	in	number	Identifies the session. If this is Null, then the procedure generates a number and returns it in x_run_id.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

REFRESH_SINGLE_MODEL

The REFRESH_SINGLE_MODEL procedure can be used to refresh a model imported from Oracle Bills of Materials (BOM).

Considerations Before Running

Before running the REFRESH_SINGLE_MODEL procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE refresh_single_model(p_api_version      IN  NUMBER,
                               p_devl_project_id  IN  VARCHAR2,
                               x_run_id           OUT NOCOPY NUMBER,
                               x_status           OUT NOCOPY NUMBER);
```

The following table describes the parameters for the REFRESH_SINGLE_MODEL procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the REFRESH_SINGLE_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.

Parameter	Mode	Data Type	Note
p_devl_project_id	in	varchar2	Required. The ID of the Model for which to refresh imported data. See Query for Models and Folders, page 18-6 for a query that provides this ID (DEVL_PROJECT_ID).
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

REFRESH_UI

The REFRESH_UI procedure refreshes an existing user interface based on the current model data. This procedure operates only on legacy Configurator User Interfaces (DHTML or Java applet) of the type generated with the limited edition of Oracle Configurator Developer.

See also:

- CREATE_UI, page 18-14
- REFRESH_JRAD_UI, page 18-34

Considerations Before Running

Before running the REFRESH_UI procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Restrictions and Limitations

This procedure only refreshes the UI specified. Referenced user interfaces are not refreshed if the specified UI is DHTML. If the referenced UI is one that is based on the OA Framework, then referenced user interfaces are refreshed.

Alternatives

As an alternative to using this procedure, you can refresh a UI in the limited edition of Oracle Configurator Developer. For more information see the Oracle Configurator Release Notes for this release.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE (p_api_version IN     NUMBER,
                     p_ui_def_id   IN OUT NOCOPY NUMBER,
                     x_run_id OUT NOCOPY NUMBER,
                     x_status OUT NOCOPY NUMBER);
```

The following table describes the parameters for the REFRESH_UI procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the REFRESH_UI Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_ui_def_id	in/out	number	UI definition ID of user interface to be refreshed. If user interface is Applet style, then a new ui_def_id is returned through this parameter. If the style is DHTML, then the same ui_def_id is returned.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored.
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_WARNING or G_STATUS_SUCCESS.

REFRESH_JRAD_UI

The REFRESH_JRAD_UI procedure refreshes an existing user interface based on the current Model data. This procedure generates only User Interfaces based on the OA Framework. For more information on the OA Framework, see the Oracle Application Framework Documentation Resources, Release 12, on the Oracle Support Web site.

See also:

- CREATE_JRAD_UI, page 18-17
- REFRESH_UI, page 18-33

Considerations Before Running

None

Alternatives

As an alternative to using this procedure, you can refresh a UI in Oracle Configurator Developer, in the User Interface area of the Workbench. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE refresh_jrad_ui(p_api_version     IN     NUMBER,
                          p_ui_def_id       IN OUT NOCOPY NUMBER,
                          x_return_status   OUT NOCOPY VARCHAR2,
                          x_msg_count       OUT NOCOPY NUMBER,
                                  OUT NOCOPY VARCHAR2);
```

The following table describes the parameters for the REFRESH_JRAD_UI procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the REFRESH_JRAD_UI Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9 .
p_ui_def_id	in/out	number	Identifies the UI to refresh
x_return_status	out	varchar2	Either G_STATUS_ERROR, G_STATUS_SUCCESS, or G_STATUS_WARNING.
x_msg_count	out	number	The number of error messages returned in the `x_msg_data` parameter.
x_msg_data	out	varchar2	A string that contains any error messages.

REPOPULATE

The REPOPULATE procedure iterates through all Populators associated with the input model and repopulates them.

Considerations Before Running

Before running the REPOPULATE procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, you can repopulate the Model with current data when data in the Item Master changes in Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE repopulate(p_api_version IN NUMBER,
                     p_devl_project_id IN NUMBER,
                     p_regenerate_all  IN  VARCHAR2 , -- DEFAULT '1',
                     p_handle_invalid  IN  VARCHAR2 , -- DEFAULT '1',
                     p_handle_broken   IN  VARCHAR2 , -- DEFAULT '1',
                     x_run_id          OUT NOCOPY NUMBER,
                     x_status          OUT NOCOPY NUMBER);
```

The following table describes the parameters for the REPOPULATE procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the REPOPULATE Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_devl_project_id	in	number	The ID of the Model to repopulate. See Query for Models and Folders, page 18-6 for a query that provides this ID (DEVL_PROJECT_ID).
p_regenerate_all	in	varchar2	Set to 0 if all Populators should be regenerated unconditionally before execution. Set to 1 to regenerate only modified Populators. The default is 1.

Parameter	Mode	Data Type	Note
p_handle_invalid	in	varchar2	Allows caller to specify how to handle invalid Populators. Pass 0 to skip invalid Populators, or pass 1 to regenerate them. The default is 1.
p_handle_broken	in	varchar2	Allows caller to specify whether to continue (1) or not (0) when a Populator cannot be regenerated successfully. The default is 1.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

REPUBLISH_MODEL

The REPUBLISH_MODEL procedure is the server side API to create a publication request and republish the model.

Only valid publications can be republished. A valid publication's DELETED_FLAG=0, STATUS=OK, and SOURCE_TARGET_FLAG=S.

Possible reasons for the REPUBLISH_MODEL procedure to fail, are:

- Input dates were not valid for the p_publication_id
- There is an overlap with existing publications for the same Model
- The Model was regenerated and the UI was refreshed

If the validation fails for any reason, the error messages are logged in CZ_DB_LOGS.

Considerations Before Running

Before running the REPUBLISH_MODEL procedure, you must first run `fnd_global.APPS_INITIALIZE` procedure. This procedure sets up global variables and profile values in a database session. Call this procedure to initialize the global security context for a database session.

Alternatives

As an alternative to using this procedure, you can republish an existing model in Oracle

Configurator Developer in the Publications area of the Repository. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this procedure is:

Example

```
PROCEDURE republish_model(p_api_version IN NUMBER,
                     p_publication_id IN NUMBER,
                     p_start_date IN DATE,
                     p_end_date IN DATE,
                     x_run_id OUT NOCOPY NUMBER,
                     x_status OUT NOCOPY NUMBER);
```

The following table describes the parameters for the REPUBLISH_MODEL procedure. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the REPUBLISH_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9.
p_publication_id	in	number	Required. This is the ID of the publication that is being republished.
p_start_datel	in	date	This is the start date of the original publication.
p_end_date	in	date	This is the end date of the original publication.
x_run_id	out	number	The ID of the running of this procedure. This value is stored in CZ_DB_LOGS.RUN_ID. If there are no warnings or errors, then 0 is stored.
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_SUCCESS, or G_STATUS_WARNING.

RP_FOLDER_EXISTS

The RP_FOLDER_EXISTS function checks whether a specified folder already exists in the Repository of Oracle Configurator Developer. You can use this function before you use CREATE_RP_FOLDER, page 18-12, to avoid trying to create a folder with a conflicting name.

This function returns the values listed in the tableValues Returned by RP_FOLDER_EXISTS, page 18-39, given the conditions shown.

Values Returned by RP_FOLDER_EXISTS

Enclosing folder (`p_encl_folder_id`)	Target folder (`p_rp_folder`)	Function Returns ...
Null	Exists anywhere in the Repository	TRUE
Not null and exists anywhere in the Repository	Exists inside enclosing folder.	TRUE
Null	Does not exist anywhere in the Repository	FALSE
Not null and does not exist anywhere in the Repository	N/A	FALSE
Not null	Does not exist inside enclosing folder.	FALSE

See also:

- CREATE_RP_FOLDER, page 18-12

Considerations Before Running

None

Alternatives

As an alternative to using this procedure, you can search for the target folder in Oracle Configurator Developer, by expanding some or all folders in the Repository. See the *Oracle Configurator Developer User's Guide* for details.

Syntax and Parameters

The syntax for this function is:

Example

```
FUNCTION rp_folder_exists (p_api_version    IN NUMBER
                          ,p_encl_folder_id IN NUMBER
                          ,p_rp_folder_id   IN NUMBER) RETURN BOOLEAN;
```

The following table describes the parameters for the RP_FOLDER_EXISTS function. This includes mode (in or out), the data type, and a brief note about the parameter.

Parameters for the RP_FOLDER_EXISTS Function

Parameter	Mode	Data Type	Note
p_api_version	in	number	Required. See API Version Numbers, page 18-9 .
p_encl_folder_id	in	number	Required. The ID of the enclosing (parent) folder containing the target folder name. To determine the ID of a folder, see Querying for Model and Folder IDs, page 18-5. To specify the root folder of the Repository, use the constant RP_ROOT_FOLDER.
p_rp_folder_id	in	number	Required. The ID of the folder that is the target of your search. To determine the ID of a folder, see Querying for Model and Folder IDs, page 18-5.

Part 5

Runtime Configurator

This Part presents information for deploying a runtime Oracle Configurator that is embedded in a host Oracle Application or a custom host application as described in Deployment Tasks, page 1-7.

19

User Interface Deployment

This chapter describes the activities required to complete the User Interface deployment of a runtime Oracle Configurator embedded in a host Oracle Application such as Order Management or *i*Store.

This chapter covers the following topics:

- Overview
- Calling an Embedded Oracle Configurator

Overview

Deployment involves making a runtime Oracle Configurator available to end users. This chapter describes the types of User Interfaces that may be deployed in a runtime Oracle Configurator.

Oracle Configurator can be deployed in these scenarios:

- Embedded in a host Oracle Application such as Order Management, using either a User Interface generated in Configurator Developer or the Generic Configurator User Interface.
- Embedded in a host application outside of Oracle Applications using a User Interface generated in Configurator Developer.
- Embedded in a host application outside of Oracle Applications using an entirely custom-written user interface that accesses the Configuration Interface Object (CIO). This scenario is not described directly in any Oracle Configurator documentation.

  The CIO and its basic usage is described in the *Oracle Configurator Extensions and Interface Object Developer's Guide*.

Calling an Embedded Oracle Configurator

Oracle Applications uses an internet server, such as Oracle Application Server, to run the Oracle Configurator (OC) Servlet. The OC Servlet connects the runtime Oracle Configurator's URL to the CZ schema. The Oracle Configurator's URL is set by the profile option BOM: Configurator URL of UI Manager.

See the *Oracle Configurator Installation Guide* for information about installing the OC Servlet and configuring the internet server.

An Oracle Configurator embedded in Oracle Applications uses one of the following user interfaces:

- A simple, non-customized UI that shows only BOM items.

  For details, see Generic Configurator User Interfaces, page 19-2.

- A customized HTML UI that is generated and optionally customized in Configurator Developer.

  For more information, see the *Oracle Configurator Developer User's Guide*.

For information about activities required to complete deployment of a runtime Oracle Configurator embedded in a host Oracle Application such as Order Management or *i* Store, see Deployment Considerations, page 20-1.

See Database Uses, page 3-2 for an overview of possible deployment environments and architecture.

Generic Configurator User Interfaces

A Generic Configurator User Interface can be accessed by host applications that are part of the Oracle E-Business Suite to configure a BOM Model. Examples of Oracle E-Business Suite host applications include Order Management, Bills of Material, Quoting, and *i*Store.

Generic Configurator UIs are not User Interfaces that are created in Oracle Configurator Developer. These UIs display only BOM Model items and enforce only implicit BOM rules. In other words, any Model structure nodes, rules, or UI elements that are defined in Configurator Developer are not available in a Generic Configurator UI. This is because Generic Configurator UIs access BOM Model data directly from the Oracle Bills of Material database tables, not from the CZ schema.

Deploying a configuration model that is based on a BOM Model and uses rules defined in Configurator Developer typically involves creating a UI in Configurator Developer and then publishing both the configuration model and the UI. For details, see the *Oracle Configurator Developer User's Guide*.

You may want your end users to use a Generic Configurator UI to configure a BOM Model item if:

- Your end users do not need a UI that provides unique selection controls, company-specific logos, custom images, and so on (for example, internal order entry employees or sales representatives).

- The BOM Model does not require additional structure or rules to support guided buying or selling questions (that is, structure and rules defined in Configurator Developer).

Criteria for Launching a Generic Configurator User Interface

A Generic Configurator UI is used when an Oracle E-Business Suite host application sends a request to configure:

- A BOM Model item that has not been imported into Configurator Developer.

- A BOM Model item that has been imported into Configurator Developer, but has not been published.

- A BOM Model item for which no matching publication is found.

> **Note:** If the host application sends a request to configure a Model that was created in Configurator Developer and no matching publication is found, Oracle Configurator displays an error.

Generic Configurator UI Types

The available types of Generic Configurator UIs are the HTML Hierarchical Table UI and the Java Applet UI. The HTML Hierarchical Table UI appears in a Web browser, is based on the Oracle Applications Framework, and is available from both Oracle Forms-based and HTML-based host applications. This UI appears when the profile option CZ: Generic Configurator UI Type is set to `HTML Hierarchical Table` and the item being configured meets the criteria described in Criteria for Launching a Generic Configurator User Interface, page 19-3. In this UI, the BOM Model is presented in a hierarchical table and controls are provided to expand and collapse configurable items, select options, and enter a quantity for each option. For more information, see Generic Configurator User Interfaces: Additional Features and Limitations, page 19-4.

The Java Applet UI does not run in a Web browser and it is available only from Forms-based host applications, such as Oracle Order Management. The Java Applet UI appears when all of the following are true:

- The host application is Forms-based

- The profile option CZ: Generic Configurator UI Type is set to `Java Applet` (see Setting Up a Generic Configurator User Interface, page 19-4)

- The item being configured meets the criteria described in Criteria for Launching a

Generic Configurator User Interface, page 19-3

The Java Applet UI contains three regions. The region on the left displays the BOM Model's hierarchical structure and enables the end user to navigate to each configurable component. End users use the region at the top of the screen to select options. The region at the bottom of the screen displays a summary of all selected options and the status of the configuration. For more information, see Generic Configurator User Interfaces: Additional Features and Limitations, page 19-4.

For more information about Forms-based applications, see the *Oracle E-Business Suite User's Guide*.

Setting Up a Generic Configurator User Interface

The following profile options modify the behavior and appearance of the HTML Hierarchical Table UI:

- CZ: BOM Tree Expansion State
- CZ: Generic Configurator UI Max Child Rows
- CZ: Hide Focus in Generic Configurator UI

By default, Forms-based host applications such as Oracle Order Management use the Java Applet UI to configure items that meet the criteria described in Criteria for Launching a Generic Configurator User Interface, page 19-3. For details about the Java Applet UI, see Generic Configurator UI Types, page 19-3.

BOM Models can contain Items that support decimal quantities and some Items may have a default quantity that is a decimal value. To configure such a BOM Model using the Generic Configurator UI, the profile option CZ: Populate Decimal Quantity Flags must be set to `Yes`. For UIs created in Configurator Developer, this profile option determines whether the BOM Model supports decimal quantities when it is imported into the CZ schema, not when the UI is launched from a host application.

If your host application is either Oracle *i*Store and Oracle Quoting, verify that the profile option CZ: Use Generic Configurator UI is set correctly for your installation.

For more information about any of the profile options referred to in this section, see the *Oracle Configurator Installation Guide*.

Generic Configurator User Interfaces: Additional Features and Limitations

The Generic Configurator User Interfaces:

- Can display pricing and Available To Promise (ATP) information (if implemented).

  To set up pricing and ATP, see Pricing and ATP in Oracle Configurator , page 13-1.

- Enable end users to search for items based on the item name or description

After the end user searches for an item in the HTML Hierarchical Table UI, the following columns are available: View in Hierarchy and Path. The View in Hierarchy column provides an icon that enables an end user to navigate directly to the item. The Path column indicates the item's location in the Model using item descriptions. For example:

```
Premium Custom Laptop Model.Hard Drive Option Class.40 GB
Hard Drive
```

- Identify unsatisfied items and items that are required to create a valid configuration
- Provide multiple languages support (MLS)
- Support secure sockets layer (SSL)
- Display currency in the same format as the host application

The Generic Configurator User Interfaces do not support:

- Multiple instantiation (creating multiple instances of configurable components)
- Connectivity (connecting configurable components)

In other words, an Oracle Configurator end user can connect and create multiple instances of configurable components only in User Interfaces that are created in Configurator Developer.

For more information about multiple instantiation and Connectivity, see the *Oracle Configurator Developer User's Guide*.

Keyboard Access in the Runtime Configurator

Oracle Configurator Developer enables end users with disabilities to navigate the runtime Configurator window using only the keyboard. For information on the available keystrokes and the corresponding actions at runtime, see the *Oracle Configurator Developer User's Guide*.

20

Deployment Considerations

This chapter describes the strategies you should consider when you are ready to complete the deployment of a runtime Oracle Configurator.

This chapter covers the following topics:

- Overview
- Deployment Strategies
- Architectural Considerations
- Server Considerations
- Establishing End User Access
- Determining the Runtime User Interface
- Load Balancing and Secure Sockets Layer
- Network Considerations
- Security Considerations
- Multiple Language Support Considerations
- Performance Considerations
- Routing Models to Specified JVMs

Overview

This chapter and User Interface Deployment, page 19-1 describe activities required to complete deployment of a runtime Oracle Configurator embedded in a host Oracle Application such as Order Management or *i*Store. The activities include:

- Deployment Strategies, page 20-2
- Architectural Considerations, page 20-2

- Server Considerations, page 20-3
- Establishing End User Access, page 20-5
- Determining the Runtime User Interface, page 20-5
- Load Balancing and Secure Sockets Layer, page 20-6
- Network Considerations, page 20-6
- Security Considerations, page 20-7
- Multiple Language Support Considerations, page 20-9
- Performance Considerations, page 20-10

Additionally, see Database Uses, page 3-2 for an overview of possible deployment environments and architecture.

Deployment Strategies

No single factor is likely to make your deployment succeed. A successful deployment depends on the relationship and interaction of several critical factors that are mentioned in this chapter.

This chapter describes the principles that affect a typical Oracle Configurator deployment.

Architectural Considerations

The architecture of an application often limits its operation. An inefficient configuration model design cannot overcome the limitation by simply tuning your server software of augmenting your hardware.

Model loading and data access depend on how the application was implemented. To get the information required to start tuning your servlet requires you to understand the application. You need to take the time to plan a model of what steps end users will experience and what variety of options will be presented, such as:

- What users select page by page
- How users navigate from page to page
- What interruptions can occur during a configuration session (for example, when a user pauses a long time to consider their choices, or turns to another task before returning to make a selection)

Server Considerations

A critical factor in deploying Oracle Configurator on your internet server is the number of instances of the servlet engine (Apache JServ) that you deploy. This number is based on the number of end users that you expect to be conducting simultaneous configuration sessions in each instance, and the kind of data access that they are going to experience.

You need to consider these factors in determining the load balance of users per JServ:

- Network data access calls made by your application
- The length of time that a user requires to work through the application
- The number of times a user can work through the application in an hour
- How many of this type of user can use your application at the same time without interfering with other users needing to access the database (for instance, to save a configuration)

Consequently, the architecture of your application affects your ability to balance the load on your server, which determines the server resources that your application requires.

The factors that affect the number of users per JServ include:

- The size of the application (the number of pages or screens)
- The size of the Model (the number of nodes)
- The number or complexity of any Configurator Extensions used by the application
- The number of CPUs
- The memory per CPU

  The JDK uses about 16 megabytes. The JVM for each JServ uses about 45 megabytes. Oracle Configurator uses native threads.
- The number of JServ instances running
- The number of connections available in the connection pool (see Connection Pooling, page 20-4)

Example

Consider a hypothetical deployment that includes:

- 6 CPUs

- 2 JServ instances per CPU
- 20 end users expected per JServ

This deployment can support 240 simultaneous user configuration sessions:

6 CPUs x 2 JServs per CPU x 20 users per JServ = 240 users

Due to the nature of the application, and the kind of data access that occurs in the application, you should consider what kind of peak events might occur when several users perform a "save" operation in the same minute.

If there are not enough database connections in the connection pool when many users save their configuration at the same time, those users will experience an unacceptable wait until enough connections are freed.

The Oracle Applications Java Caching Framework (OAJCF) is the caching mechanism for all Oracle Applications products. This mechanism stores database results and Java objects in memory for repeated usage, thus minimizing expensive object initializations and database round-trips as well as improving application performance. Consider setting up a dedicated JServ for running Oracle Configurator.

For more information, see the *Oracle Configurator Performance Guide*.

Connection Pooling

Connection pooling allows multiple configuration sessions in a JServ instance to make database connections. (Previous versions of Oracle Configurator were only able to use a single database connection for each JServ instance.)

When a configuration session is started by the posting of the initialization message to the OC Servlet, a connection is obtained from the pool. When the session is over, the connection is returned to the pool. Each connection requires memory.

Oracle Configurator uses AOL/J (Java classes for AOL (Applications Object Library)) to provide connection pooling. To modify the default setting for connection pooling, you use the AdminAppServer class to create or update a DBC file, setting a value for the parameter FND_MAX_JDBC_CONNECTIONS.

The parameter FND_MAX_JDBC_CONNECTIONS specifies the maximum number of open connections in the JDBC connection cache. This number is dependent on the amount of memory available, the number of processes specified in the `init.ora` file of the database, and the per-processor file descriptor limit.

The maximum pool size is the maximum allowed sum of the number of available connections and the number of locked connections. If the .dbc file does not have a setting for maximum pool size, the default value is used. The default value is the Java static field Integer.MAX, which normally has a value of about 2 billion. Therefore, the default value is essentially unlimited.

The parameter FND_JDBC_MAX_WAIT_TIME specifies the length of time a request waits for a connection to be established. The default value is 10 seconds, and this

parameter is not configurable.

Establishing End User Access

End users ability to access the runtime Oracle Configurator are established by the Oracle Applications System Administrator. For more information, see the *Oracle E-Business Suite System Administrator's Guide*. For more information about the behavior of the runtime Oracle Configurator as it affects end users, see the *Oracle Configurator Developer User's Guide*.

Publication applicability parameters also affect end-user access to configuration models. For example, the effective dates and times of the configuration model publication must be valid for the time setting on the computer where the host application is running. For more information about publication applicability parameters, see Determining the Runtime User Interface, page 20-5, and the *Oracle Configurator Developer User's Guide*.

Determining the Runtime User Interface

The settings of a Model publication's applicability parameters, the initialization message sent by the host application, and the end user's responsibility determine which type of user interface is displayed in a runtime Oracle Configurator. For more information, see User Interface Deployment, page 19-1.

For example, an end user is expecting to see a generated Configurator UI at runtime but instead sees the Generic Configurator User Interface. This can happen when the host application is not specified in the publication's applicability parameters, or the end user's responsibility is not valid for the host application. For details about the Generic Configurator UI, see Generic Configurator User Interfaces, page 19-2.

To determine what responsibilities are valid for an application, two queries can be run. By querying the local database with the specified application short name, the application ID can be retrieved and then used in a second query to determine what responsibility IDs are valid for the specified application ID.

Example

```
SELECT application_id, application_short_name, description
FROM FND_APPLICATION_Vl
WHERE application_short_name='CSS'
APPLICATION_ID      APPLICATION_SHORT_NAME    DESCRIPTION
514                 CSS                       Support
```

Using the returned `APPLICATION_ID` you can then run another query to determine the responsibilities that are allowed for that application:

```
SELECT application_id, responsibility_id, responsibility_name,
responsibility_key
FROM fnd_responsibility_VL
WHERE application_ID = '514'
APPLICATION_ID  RESPONSIBILITY_ID  RESPONSIBILITY_NAME
RESPONSIBILITY_KEY
     514          12345            Customer Support Test
Oracle_Support
     514          67890            Customer Support USA
Customer_Support
```

For information about legacy UIs, see the current release or patch information for Oracle Configurator on MetaLink, Oracle's technical support Web site.

Load Balancing and Secure Sockets Layer

Oracle strongly recommends using Oracle Application Server 10*g*. This version can be set up to use a process manager that automatically load balances server processes and supports Secure Sockets Layer (SSL) for greater security when transmitting data over the Internet.

Refer to Oracle Application Server documentation. For additional SSL information, see Oracle Applications Documentation, on the Oracle Technology Network.

If you are not using Oracle Application Server, refer to the following Apache Web sites for more information about load balancing and SSL:

Example

```
http://java.apache.org/jserv/howto.load-balancing.html
http://www.apache-ssl.org
```

Network Considerations

There are a number of network issues that can cause serious problems for your deployment if not handled correctly.

Firewalls and Timeouts

If your application requires more than one server system, then it is recommended that there be separate servers for the Oracle database server and the internet server. If there are firewalls between servers, then these firewalls must allow persistent database connections between them. Persistent database connections are SQL links that do not close when the execution of your script ends.

The OC Servlet is a stateful application. A stateful application keeps its critical data in memory, rather than writing and reading it from disk storage. Oracle Configurator keeps in memory critical data, such as the Properties cache and the state of the logic engine, until a configuration is saved.

Stateful applications require a persistent connection between the database server port and the ports used by the servlet engine (in this case, Apache JServ). The default timeout for the JServ engine is 30 minutes.

Warning: If there is an idle time limit set on the TCP/IP database connection across a firewall, then this limit can prevent Oracle Configurator from operating.

See Security Considerations, page 20-7 for firewall recommendations.

Router Timeouts

Routers have a setting referred to as "stickiness." Router stickiness connects the HTTP request made by a particular client browser (that is, the browser displaying the runtime Oracle Configurator) with a particular instance of the servlet engine (JServ).

The stickiness setting is a time limit on the total time allowed for the connection between client and servlet engine. After the time limit is exceeded by the client browser, the connection to the servlet engine instance is broken. If the end user attempts to use the browser, then it is possible that the router may connect that browser to a *different*, and wholly incorrect, servlet engine instance.

You must determine the appropriate length of time for your application. For instance, if you feel that your users may wish to use your application for one hour, then you must set the router stickiness to match that time.

Warning: If the "stickiness" time limit of your router is too small for the correct use of your application, this limit can prevent Oracle Configurator from operating.

See Security Considerations, page 20-7 for router recommendations.

Miscellaneous Issues

- Your application must run in an environment that resolves domain names to allow it to communicate with other servers.
- You must set up your router and server so that all users and processes have the access privileges and permissions they need in order to carry out their functions.

Security Considerations

If you are implementing Oracle Configurator outside your firewall, then consider the following recommendations:

- Protect your servers with a firewall.
- Have an additional firewall between the application server and the database server. This additional firewall can guard against unauthorized access to the database

server. It should be configured to open only designated ports for application server access to the database.

Additional servers intended for internal use should also be behind this firewall.

- Use hardware routers and front-door products like Oracle's WebCache to provide an additional level of security.

- Use separate computers or clusters for the application server and the database server. This is always recommended for performance reasons, but in the context of security it also provides the benefit of preventing a denial-of service attack from disabling the database server.

Some risks still remain in that a malicious user could gain access to the application server. Oracle recommends the following:

- Dedicate a computer or cluster to the public Web site's application server. This will minimize the functionality to which a malicious user would have access. This server should not mount sensitive file systems and should be isolated from the internal servers by a firewall.

- Do not store database connection parameters (for example, .dbc files) that provide extensive database access on the same application server that is used for public Web site access. For more information on database connections, see Server Considerations, page 20-3 and Internet User Access , page 20-8.

- Disable default database account and Oracle Applications users that ship with Oracle products.

Internet User Access

There is no direct database connection from non-authenticated Internet users to your production database because:

- The database connection is established by the Configurator middle tier that is running on the application server. The connection is not established by any software running on the client's computer.

- Database connection parameters are secured on the application server using AOL/J functionality based on Oracle Applications FND authentication. For more information, see Oracle Applications Documentation, on the Oracle Technology Network.

- Database connection information is not transmitted over the Internet. An encrypted ICX session ticket that is valid just for a single application server session is transmitted. For more information on an ICX session ticket see Configurator Architecture, page 2-1, and the *Oracle Configurator Extensions and Interface Object Developer's Guide*.

- Application server sessions for a public Web site logs into Oracle Applications with an Oracle Applications user ID assigned for walk-in users. The walk-in user is defined to have a valid Oracle Applications responsibility that provides access only to the necessary functions. The walk-in user will not have database login privileges. For additional access, see Establishing End User Access, page 20-5.

Additional Security Precautions

The following security precautions may also be considered if your public Web site does not require live access to production data such as entering orders or updating account information:

- Use a second Oracle Applications instance to host the implementation of the runtime Oracle Configurator if the runtime Oracle Configurator does not require data from any part of Oracle Applications other than the CZ schema.

- Application server sessions for the public Web site connect to the runtime Oracle Configurator database instance, not the production database instance. Database access from the public application server to the production database instance is not available.

- Create and maintain configuration models in the production database instance, and then publish the Models to the runtime Oracle Configurator database instance. Any custom data that is needed for the public Web site would need to be stored or duplicated on the runtime Oracle Configurator instance.

- If there are any transactions that a consumer could start through the public Web site, then you would have to implement a procedure to extract the transactional data from the runtime Oracle Configurator database instance and import it to the production database instance for processing. This extraction is not necessary if the only output of the public Web site is information for the consumer.

- If feedback on the state of transactions in the production database instance must be provided to end users on the public Web site, then you have to implement a procedure to extract this data from the production database instance and import it into the runtime Oracle Configurator database instance. This data would only be as current as of the most recent extraction.

Multiple Language Support Considerations

If you are implementing Multiple Language Support (MLS), see Multiple Language Support, page 14-1.

Performance Considerations

For information about improving the performance of your runtime Oracle Configurator, see the *Oracle Configurator Performance Guide*.

Routing Models to Specified JVMs

To improve performance and reduce memory usage, you can route specific Models at runtime to specific JVMs.

Solution Overview

By default, Oracle Configurator caches all Models in a session on every Java Virtual Machine (JVM) running on the web server. If you have very large Models, you may encounter performance problems related to the large memory footprint of those Models. Caching of large Models can overload the JVMs, eventually hitting the maximum heap limit, which in turn causes JVMs to die.

To circumvent this situation, you can route specific Models only to specific pools of JVMs, thus reducing the memory footprint load on the other JVMs. This solution requires the use of a Load Balancing Router (LBR), which is a hardware item that distributes HTTP requests to the servers in a server farm based on load-balancing methods such as round-robin or least-load. An LBR can read the HTTP information of the traffic to and from the web servers connected to it.

To augment the pool-routing capability of an LBR, Oracle Configurator exposes the pool-specifying information of a Model when the Model is launched at the beginning of a configuration session. The pool information for a given Model is placed in the HTTP response message, enabling the LBR to read the pool information for the Models from the HTTP traffic during a configuration session and thus route HTTP request messages to the desired JVM pool.

This routing capability requires a combination of the following elements:

- Changes in your LBR routing rules to read a new pool token cookie, `czPoolToken`, from the incoming cookies for each HTTP request and accordingly route the request to a specified JVM pool.

- Creation of mappings between JVM pools and Oracle Configurator Models, which you insert in the table CZ_MODEL_POOL_MAPPINGS using the procedure CZ_CF_API .REGISTER_MODEL_TO_POOL, page 17-63, to specify the Models to be routed to specified pools. The table includes the rows MODEL_PRODUCT_KEY and POOL_IDENTIFIER to store the mappings.

Efficient routing rules and mappings can improve performance and reduce the memory footprint in each JVM.

Solution Processing

The general processing in this solution is as follows:

- The profile option CZ: Add Model Routing Cookie must be set to True for Model routing to occur.
- The runtime Oracle Configurator launches a Model via the initialization message.
- Oracle Configurator looks up the publication used for the Model and obtains a Product Key that identifies it.

  (The Product Key for a Model appears in the General area of the Workbench in Oracle Configurator Developer. For BOM Models, its value is a combination of the Inventory Item ID and Inventory Organization ID, such as 204:143. See the *Oracle Configurator Developer User's Guide* for more details about Product Key. The Product Key is the same as the Product ID, page 16-7 used when publishing Models.
- Oracle Configurator uses the Product Key to obtain the name of the intended JVM pool for the Model from the table CZ_MODEL_POOL_MAPPINGS.
- Oracle Configurator adds an HTTP cookie named `czPoolToken` to the HTTP response message from the server. The cookie contains the name of the JVM pool for the Model being configured.
- Subsequent HTTP requests include the pool token, which enables the LBR to route the request to the intended JVM pool. The pool token cookie is destroyed when the end user exits the configuration session by selecting Finish or Cancel.

Solution Procedure

To implement this model routing solution:

1. Set the profile option CZ: Add Model Routing Cookie to True.

   This enables the addition of the routing cookie to the HTTP response content. By default, the value is False. This profile option can be set only at Site level. See the *Oracle Configurator Installation Guide* for details.

2. Register your Models to the JVM server pools that should serve requests to configure them. To do this, run the procedure CZ_CF_API .REGISTER_MODEL_TO_POOL, page 17-63.

   Example: The following simplified example registers the Model with Product Key `204:143` to the JVM pool `PoolA`:

   ```
   cz_cf_api .register_model_to_pool('PoolA', '204:143')
   ```

   You can examine your mappings with the following SQL query:

```
select MODEL_PRODUCT_KEY, POOL_IDENTIFIER from
CZ_MODEL_POOL_MAPPINGS;
```

The query might produce output like the following example of mappings for several Models and pools:

```
MODEL_PRODUCT_KEY  POOL_IDENTIFIER
-----------------  ------------------------
204:143            PoolA
204:137            PoolB
204:19886          PoolC
```

3. Modify your LBR routing rules so that they read the new pool token in the cookie named `czPoolToken` and route the Configurator requests to the correct JVM pools.

   Example: The following rule for the BIG-IP LBR routes an Oracle Configurator pool token for the CZ mapping table identifier `PoolA` to the LBR pool identified in the LBR rules as `pool_x`, which must be previously defined for the LBR:

   ```
   if ( exists http_cookie ("czPoolToken") and http_cookie
   ("czPoolToken ") contains "PoolA")
   { use pool pool_x }
   else { use pool pool_y }
   ```

4. Continue with the procedures that are specific to your LBR for putting the LBR rules into effect.

21

Managing Configurations

This chapter describes the data structures produced by Oracle Configurator during a configuration session, and how to manage the life cycle of a configuration.

This chapter covers the following topics:

- Overview
- About Configurations
- Configuration Identity
- Host Applications and Oracle Configurator
- Batch Validation of a Configured Item
- Reconfiguring a Configured Item
- Copying a Host Application's Entity
- Passing a Saved Configuration to Another Host Application
- Deleting a Host Application Entity

Overview

This chapter explains the data structures produced by Oracle Configurator during a configuration session and how to manage the lifecycle of saved configurations. It includes the following topics:

- Copying a Host Application's Entity, page 21-7
- Passing a Saved Configuration to Another Host Application, page 21-8
- Deleting a Host Application Entity, page 21-8

For general information, see Configurator Architecture, page 2-1. For related information about configuration models and rules, and about the behavior of the runtime Oracle Configurator, see the *Oracle Configurator Developer User's Guide*.

About Configurations

A configuration is the record of a configuration session. It is the output produced by the runtime Oracle Configurator, as a product of processing an end-user's selections, which cause configuration rules to be applied against a configuration model. Oracle Configurator validates the selections, resulting in a configuration.

Once a configuration has been saved during a configuration session, it is identified by a configuration header ID, which is stored in the CZ schema as CZ_CONFIG_HDRS.CONFIG_HDR_ID.

When a configurable item is successfully configured, the `config_hdr_id` and `config_rev_nbr` that is returned in the XML termination message should be stored in the application entity that is associated with the configured item. For example, in Oracle Order Management, this is stored on the order line. In Oracle Order Capture, it is stored on a quote line.

A configuration can be:

- Valid or invalid

  A valid configuration contains no contradictions to the rules, whereas an invalid configuration contains contradictions.

- Complete or incomplete

  A complete configuration includes all the required selections. An incomplete configuration lacks some required selections; in other words, some of the configuration rules are unsatisfied.

- New, saved, restored, or cancelled

  A new configuration is one in which the user has not made any selections, and the logic state of many elements is Unknown.

Saving a Configuration

At any time during a configuration session the configuration can be saved, thus recording the selections made against the nodes of the Model structure, which are called configuration inputs. A configuration does not have to be valid or complete in

order to be saved. You can save any configuration, even if it is invalid and incomplete. The saved configuration should be stored in the host application entity even if its status indicates that the configuration is invalid or incomplete.

If a configuration has been saved, then later it can be restored for further selections and validation. When a configuration is restored, it is not the final saved state of the Model that is restored, but only the configuration inputs to the Model. The restored inputs are reasserted against the Model to produce a configuration. See Configuration Identity, page 21-4 for more information.

If the configuration model or rules have changed since the configuration was saved, then validation failures may occur as a result of inputs that no longer match the Model.

Because restoring a configuration reasserts all the configuration inputs to the Model, restoring a configuration programmatically with the CIO is normally not faster than restoring a configuration interactively, and under some circumstances can be slower.

A configuration can also be canceled during a configuration session, by terminating the runtime Oracle Configurator without saving the configuration. In this case, the configuration inputs are discarded.

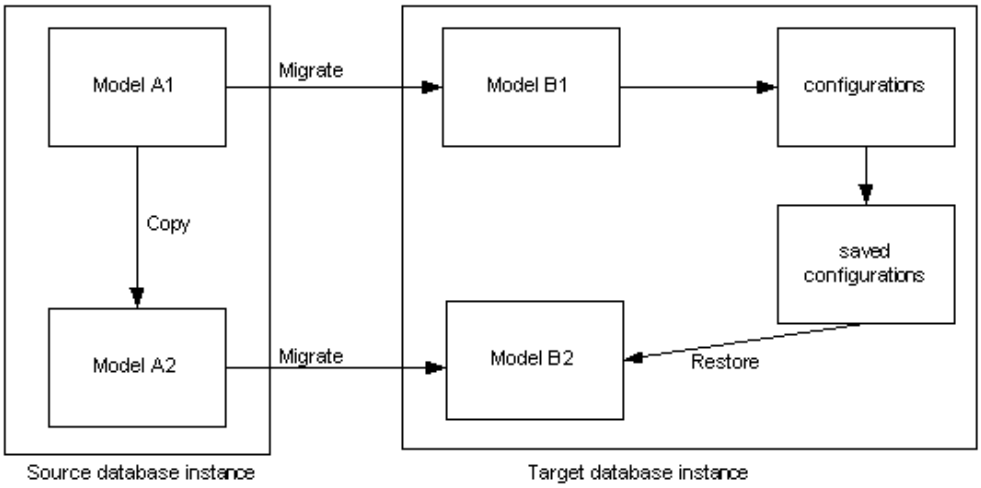
Restoring Saved Configurations

Modifying previously saved configurations are validated against a comparable Model. After a configuration is saved, the original Model's structure, rules or UI may change or be migrated to a new instance. When a Model is migrated to a new instance, the synchronization creates a comparable Model that allows the previously saved configuration to be restored and validated against.

See Synchronizing Migrated Model Data, page 6-8 for more information on synchronization criteria.

The following illustration shows what happens when a saved configuration is restored against a Model that has been migrated to and changed in another instance after the configuration was saved.

Restoring Saved Configurations Created from a Migrated Model



Model A1 is migrated to another instance as Model B1. Configurations are made against Model B1. A copy of Model A1 is made on the source instance (Model A2). At a later time Model A2 is migrated to the same target instance (Model B2). When the saved configurations are restored, they are restored against Model B2 (the most recent version of the Model).

Usually, Model node names are unique within an individual tree level and can be used when restoring configurations. Occasionally, the node name matching may fail. If you want to restore configurations that were saved against a Model that has been migrated, you must run either the Add Model Node Names to Configurations by Model Items, page C-35 or the Add Model Node Names to Configurations by Product Key, page C-37 concurrent program.

Configuration Identity

Configurations commonly consist of a single instance of your configuration model and a set of configuration inputs.

When a configuration is restored and changed, the changes are saved as a revision to that configuration. Each saved revision is identified by a Configuration Revision Number, which is stored as CZ_CONFIG_HDRS.CONFIG_REV_NBR. The combination of Header ID and Revision Number identifies a unique configuration record. The identity of each item in the configuration is recorded by a Configuration Item ID (stored as CZ_CONFIG_ITEMS.CONFIG_ITEM_ID). For detailed information about these and other tables, see the Oracle Integration Repository.

Host Applications and Oracle Configurator

Oracle Configurator does not provide a UI to manage saved configurations. Oracle Configurator is an embedded component of other applications referred to as host applications. It is the responsibility of the host application to manage saved configurations. The host application has the following responsibilities in its relationship with Oracle Configurator:

- Maintain an index of configuration product keys that can be used to launch the runtime Oracle Configurator UI or batch validation. The product key usually consists of the Inventory Item ID followed by its Inventory Organization ID. For example, 452:1534. The product key could also be any name that identifies a configurable object in the host application's domain.

- Implement the runtime Oracle Configurator UI or batch validation by providing a product key or the ID of a saved configuration. To launch a saved configuration you must know the configuration's header ID (`config_header_id`) and revision number (`config_rev_number`). For more information about the configuration's identity, see Configuration Identity, page 21-4.

- Keep track of the saved configurations returned by the runtime Oracle Configurator by storing the `config_header_id` and the `config_rev_number` with an entity in the host application.

  Note: Oracle Configurator creates saved configurations at the end of an interactive or batch configuration session when the initialization message includes instructions to do so and the session terminates normally. For more information on the initialization message, see Session Initialization, page 9-1.

- Delete saved configurations by using CZ_CF_API.DELETE_CONFIGURATION, page 17-51 when configurations are no longer associated with any host application entity.

Batch Validation of a Configured Item

Batch validation allows a host application to perform tasks such as:

- Validating a BOM-based configuration in the background

- Determining a configuration quantity

A host application calls batch validation through the CZ_CF_API.VALIDATE PL/SQL procedure. For more information on batch validation, see Batch Validation, page 11-1.

If batch validation is unsuccessful (CZ_CF_API.VALIDATE, page 17-69 returns validation_status>0), then the original config_hdr_id and `config_rev_nbr`, if any, should be preserved in the host application's entity.

If batch validation is successful (CZ_CF_API.VALIDATE, page 17-69 returns validation_status=0), then the host application must decide whether to store the returned `config_hdr_id` and `config_rev_nbr` in the host application's entity. Consider the following when storing the returned `config_hdr_id` and `config_rev_nbr`:

- If the validation is for an item that was not previously configured, then the returned `config_hdr_id` and `config_rev_nbr` should *always* be stored, because this is the original configuration of the item.

- If the validated configuration is complete and valid, then the new config_hdr_id and config_rev_nbr should be stored, replacing the previous values. The previously saved configuration should be deleted by CZ_CF_API.DELETE_CONFIGURATION, page 17-51.

- If the validated configuration is incomplete or invalid, then there are two different approaches that the host application may adopt. The host application may:

  - Choose to present the validation messages to the user and roll back whatever change in the configuration or status is being validated. In this case, the new saved configuration that is returned by batch validation should be deleted with CZ_CF_API.DELETE_CONFIGURATION, page 17-51. This is the approach that is adopted by Oracle Order Management.

  - Choose to accept any changes to the configuration, replace the previously saved configuration with the new configuration, present the validation messages to the user and roll back any proposed change in status. In this case, the previously saved configuration should be deleted with CZ_CF_API.DELETE_CONFIGURATION, page 17-51.

The key requirement is that the host application must delete whichever saved configuration that is *not* retained in the host application's entity.

Reconfiguring a Configured Item

The host application's action following the reconfiguration of a configured item depends on the value of the termination message's exit, page 10-6 element.

- If the exit value is `save`, then the termination message also contains new values for `config_hdr_id` and `config_rev_nbr`. These new values should be stored in the host application's entity that is associated with the configured item. The previously saved configuration should be deleted by calling CZ_CF_API.DELETE_CONFIGURATION, page 17-51 and passing the values of

and config_rev_nbr that were previously stored with the host application's entity.

> **Note:** This assumes that the reconfigured item replaces the previous configuration on the same host application entity. If the reconfiguration is performed in the process of creating a new copy or revision of the entity, then the new values of config_hdr_id and config_rev_nbr should be stored with the new copy or revision, and the original values should remain associated with the original entity.
>
> In this case the previously saved configuration should *not* be deleted, because it is accessible through the original host application entity.

This behavior is independent of whether the newly saved configuration is valid or complete. The user chose to save the configuration knowing its status (valid or complete), so it should be stored with the host application's entity.

- If the exit value is cancel, error, or processed, then the previously stored values of config_hdr_id and config_rev_nbr should be retained in the host application's entity.

> **Note:** Changing the Instantiability settings for a Model or Component node within a published Model may change the number of instances that exist when an end user restores a saved configuration. For example, decreasing the Initial Minimum in Configurator Developer and then republishing the Model may cause some instances of the component to be lost when the configuration is restored. (In this case, Oracle Configurator displays a message indicating that a validation failure occurred.) Similarly, increasing the Initial Minimum value may create additional instances in a restored configuration.

Copying a Host Application's Entity

When a host application creates a copy of a configuration that holds a reference to a saved configuration it should copy the saved configuration with CZ_CF_API.COPY_CONFIGURATION, page 17-37. The new config_hdr_id and config_rev_nbr that are returned from should be stored with the copy of the host application entity. The original saved configuration should *not* be deleted.

This same logic applies when the host application creates a new revision of its configuration that holds a reference to a saved configuration.

If the copied configuration must be revalidated at the time of copying, the best

approach is to use CZ_CF_API.VALIDATE, page 17-69 to create the copied configuration. Pass the parameter save_config_behavior, page 9-33=`new_config` in the initialization message, and store the `config_hdr_id` and `config_rev_nbr` to identify the copied configuration. The host application that uses this approach must be prepared to handle validation failures that may occur during the copying of a configuration.

For more information on the initialization message, see Session Initialization, page 9-1. For more information on the procedures and functions in CZ_CF_API, see Programmatic Tools for Development, page 17-1.

Passing a Saved Configuration to Another Host Application

When a saved configuration is handed off from one host application to another as part of the business flow, the saved configuration should be copied. See Copying a Host Application's Entity, page 21-7.

Assuming that the entity is still accessible in the original host application, the original host application entity should retain its reference (`config_hdr_id` and `config_rev_nbr`) to the original saved configuration. The corresponding entity in the second host application should store a reference to the copied configuration. In this case, the original saved configuration should *not* be deleted. An example of this flow is the transition from Oracle Order Capture to Oracle Order Management when a quote is submitted as an order.

Deleting a Host Application Entity

When a host application deletes, purges, or otherwise makes an entity inaccessible that holds a reference to a saved configuration, the host application should delete the configuration using the procedure CZ_CF_API.DELETE_CONFIGURATION, page 17-51.

A

Terminology

This appendix defines the terms that are found in the *Oracle Configurator Implementation Guide* that are not defined in the Glossary.

This appendix covers the following topics:

- Overview

Overview

This chapter presents terminology used in this book and not included in the Glossary of Terms and Acronyms.

The following table lists terms that are used throughout this book.

Terminology Used in This Book

Term	Description
concurrent manager	A process manager that coordinates the concurrent processes generated by users' concurrent requests. An Oracle Applications product group can have several concurrent managers.
concurrent process	A task that can be scheduled and is run by a concurrent manager. A concurrent process runs simultaneously with interactive functions and other concurrent processes.
concurrent processing facility	An Oracle Applications facility that runs time-consuming, non-interactive tasks in the background.
concurrent request	A user-initiated request issued to the concurrent processing facility to submit a non-interactive task, such as running a report.

Term	Description
ICX	Inter-Cartridge Exchange

See the Glossary, page Glossary-1 for additional terms.

B

Common Tasks

This appendix describes certain tasks that may be required while implementing an Oracle Configurator.

This appendix covers the following topics:

- Overview
- Running Configurator Concurrent Programs
- Connecting to a Database Instance
- Verifying CZ Schema Version
- Server Administration
- Viewing the Status of Configurator Concurrent Requests
- Viewing Log Files
- Managing Oracle Configurator Caching
- Checking BOM Model and Configuration Model Similarity

Overview

This appendix describes common tasks of an Oracle Configurator implementation:

- Checking BOM Model and Configuration Model Similarity, page B-4
- Managing Oracle Configurator Data Caching, page B-4

For details about specific Oracle Configurator concurrent programs, see Concurrent Programs, page C-1.

Running Configurator Concurrent Programs

To run any Oracle Configurator concurrent programs, you must log in to Oracle Applications and select one of the predefined Oracle Configurator responsibilities. For details about these responsibilities, including to which concurrent programs they provide access, see The Predefined Configurator Developer Responsibilities section in Chapter 15 of this document. For information about assigning responsibilities, see the *Oracle E-Business Suite System Administrator's Guide*.

The procedure for running concurrent programs is provided in the *Oracle E-Business Suite User Guide.*

For details about specific Oracle Configurator concurrent programs, see Concurrent Programs, page C-1.

Connecting to a Database Instance

Some implementation tasks must be performed using SQL*Plus while connected to a specific database instance. For example, during data migration you must connect to your source database instance prior to running a SQL script that sets up the migration packages, database link, and appropriate log file.

> **Note:** Connecting to a database instance using SQL*Plus is not to be confused with starting and logging on to Oracle Applications. For information on logging on to Oracle Applications, see the *Oracle Application User's Guide*.

To connect to a specific database, you must specify a user or schema and the instance in which it is defined. For example:

1. Connect to your CZ schema by connecting to the database instance as a user of the schema.

   Example

   ```
   Example:
   SQL> connect oc/ocpass@appssid
   ```

   where `oc` is the owner (DBOwner) of the CZ schema, and *`ocpass`* is the owner's password, and `appssid` is the name for the database instance.

   Alternatively, connect to the database instance as a user with DBA privileges:

Example

```
Example:
SQL> connect dba/dbapass@appssid
```

Verifying CZ Schema Version

You can determine the version information of an CZ schema by either running the View Configurator Parameters , page C-2 concurrent program or by querying the CZ_DB_SETTINGS table as follows:

1. Connect to the database instance in which you need to know the version information of the CZ schema.

2. Use SQL*Plus to enter the following query:

   Example

   ```
   SQL> select setting_id, value, desc_text
   from cz_db_settings
   where setting_id like '%_VERSION"
   ```

Querying the version of Release 12 available with the publication of this book results in MAJOR_VERSION = 27, MINOR_VERSION = a.

These values will vary depending on the latest installed version. To determine which version of the limited edition of Oracle Configurator Developer goes with which version of Oracle Configurator, refer to the Oracle Configurator Patch Matrix (Doc ID 131088.1) on the Oracle Support Web site.

For information about MAJOR_VERSION, MINOR_VERSION, and other CZ_DB_SETTINGS parameters, see CZ_DB_SETTINGS Table, page 4-10.

Server Administration

If you are using separate database instances you need to define, enable, and possibly modify the remote server. Defining and enabling a remote server establishes the database link for:

- Importing data (see Populating the CZ Schema, page 5-1)
- Publishing configuration models (see Publishing Configuration Models, page 16-1)

Oracle Configurator provides the following Server Administration concurrent programs for the Oracle Configurator Administrator responsibility in Oracle Applications:

- Define Remote Server, page C-10
- Enable Remote Server, page C-12
- Modify Server Definition, page C-14

- View Servers, page C-13

For details on running these concurrent programs, see Server Administration Concurrent Programs, page C-9.

Viewing the Status of Configurator Concurrent Requests

Any Oracle Applications reportor program that you submit is sent to the concurrent manager and is stored in a queue with other requests until it is selected for processing. You can view the status of your concurrent requests by selecting Concurrent Programs > View from the Oracle Application Navigator window. The Requests page displays the Name, Status, Phase, Scheduled Date, Request ID and other details about your requests. To view the log file for a completed concurrent program, click the icon in the Details column, and then click View Log.

For additional information, see the *Oracle Application User's Guide*.

Viewing Log Files

Log files contain error and warning messages that result from running a concurrent program or a SQL script. For information about the location of log files generated when running scripts, see *Oracle Configurator Installation Guide*. For information about viewing log files that result from running a concurrent program, see Viewing the Status of Configurator Concurrent Requests, page B-4. See Publishing Error when Checking BOM Model and Configuration Model, page 16-13 for an illustration of an error found in CZ_DB_LOGS.

Managing Oracle Configurator Caching

Oracle Applications Java Caching Framework (OAJCF) is the caching mechanism for all Oracle Applications products. OAJCF provides the ability to manage the caching and decaching of Model and UI data thereby improving performance of a runtime Oracle Configurator. For more information, see the *Oracle Configurator Performance Guide*.

Checking BOM Model and Configuration Model Similarity

See Synchronizing BOM Model Data, page 7-2 for more information on checking the similarity between the configuration model and the original BOM Model.

C

Concurrent Programs

This appendix describes the concurrent programs available to either the Oracle Configurator Administrator or Oracle Configurator Developer responsibility.

This appendix covers the following topics:

- Overview
- Configurator Administration Concurrent Programs
- Convert Publication Target Instance to Development Instance
- Server Administration Concurrent Programs
- Configuration Model Publication Concurrent Programs
- Populate and Refresh Configuration Models Concurrent Programs
- Model Synchronization Concurrent Programs
- Execute Populators in Model
- Migration Concurrent Programs
- Migrate Functional Companions
- Model Management
- Publication Synchronization Concurrent Programs

Overview

This appendix explains how to use the Oracle Configurator concurrent programs that are available to the Oracle Configurator Developer and Oracle Configurator Administrator responsibility in Oracle Applications:

For general information about concurrent requests, programs, and processes, see the *Oracle E-Business Suite System Administrator's Guide - Maintenance*.

- Configurator Administration Concurrent Programs, page C-2

- Server Administration Concurrent Programs, page C-9
- Configuration Model Publication Concurrent Programs, page C-15
- Populate and Refresh Configuration Models Concurrent Programs, page C-18
- Model Synchronization Concurrent Programs, page C-25
- Execute Populators in Model Concurrent Program, page C-29
- Migration Concurrent Programs, page C-30
- Migrate Functional Companions, page C-32
- Model Management, page C-35
- Publication Synchronization Concurrent Programs, page C-40
- Viewing the Status of a Configurator Concurrent Request, page B-4

For general information about running Oracle Configurator concurrent programs, see Running Configurator Concurrent Programs, page B-2.

Configurator Administration Concurrent Programs

The configurator administration concurrent programs are:

- View Configurator Parameters , page C-2
- Modify Configurator Parameters, page C-3
- Purge Configurator Tables, page C-4
- Purge Configurator Import Tables, page C-5
- Purge To Date Configurator Import Tables, page C-6
- Purge To Run ID Configurator Import Tables, page C-7
- Convert Publication Target Instance to Development Instance, page C-8

View Configurator Parameters

The View Configurator Parameters concurrent program allows the viewing of parameter values stored in the CZ_DB_SETTINGS table. See CZ_DB_SETTINGS Table, page 4-10 for details about the parameters in that table.

Use the procedure described in Running Configurator Concurrent Programs, page B-2

to run this concurrent program.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs** >**Schedule**

Parameters

The following table describes the parameters for the View Configurator Parameters concurrent program

Parameters for the View Configurator Parameters Concurrent Program

Parameter	Description
Section Name	From the list of values, select the SECTION_NAME of the section in the Oracle Configurator CZ_DB_SETTINGS Table, page 4-10 where the setting resides. For example IMPORT. See Settings in CZ_DB_SETTINGS Table, page 4-12 for more information.
Setting ID	From the list of values, select the SETTING_ID in the Oracle Configurator CZ_DB_SETTINGS Table, page 4-10 for the setting. For example, CommitSize, page 4-17. See for more information.

Output

The output containing the values for the specified SECTION_NAME and SETTING_ID is recorded in a log file (see Viewing the Status of a Configurator Concurrent Request, page B-4).

Modify Configurator Parameters

The Modify Configurator Parameters concurrent program allows the changing of parameter values in the CZ_DB_SETTINGS table. Configurator parameters are stored in the CZ_DB_SETTINGS table. See CZ_DB_SETTINGS Table, page 4-10 for details about the parameters in that table.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs**: **Schedule**

Parameters

The following table describes the parameters for the Modify Configurator Parameters concurrent program

Parameters for the Modify Configurator Parameters Concurrent Program

Parameter	Description
Section Name	From the list of values, select the name of the section in the Oracle Configurator CZ_DB_SETTINGS table where the setting resides. See Settings in CZ_DB_SETTINGS Table, page 4-12.
Setting ID	From the list of values, selectthe setting in the Oracle Configurator CZ_DB_SETTINGS table you want to modify. See Settings in CZ_DB_SETTINGS Table, page 4-12 for more information on the settings in each SECTION_NAME.
Value	Enter the value for the particular parameter. See CZ_DB_SETTINGS Table, page 4-10 for more information on valid values for each of the settings in each SECTION_NAME.
Type	From the list of values, select the data type (1= number or 4= string) of the setting you are modifying.
Description	Enter a brief description for this value selection.

Output

The output containing the modified values of the CZ_DB_SETTINGS Parameters, page 4-11 you specified is recorded in a log file. For details, see Viewing the Status of a Configurator Concurrent Request., page B-4

Purge Configurator Tables

The Purge Configurator Tables concurrent program physically removes all

logically-deleted records in the tables and subschemas of the CZ schema. Periodically running this concurrent program improves database performance. See CZ Schema Maintenance , page 8-1 for more information about purging the CZ schema. See the *Oracle Configurator Performance Guide* for additional information about improving database performance.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs**: **Schedule**

Parameters

None

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Purge Configurator Import Tables

The Purge Configurator Import Tables concurrent program deletes all data in the CZ_IMP tables, and the corresponding data in the CZ_XFR_RUN_INFOS, and CZ_XFR_RUN_RESULTS control tables. Periodically running this concurrent program improves import performance. See CZ Schema Maintenance , page 8-1 for more information about purging the CZ schema. See the *Oracle Configurator Performance Guide* for additional information about improving database performance.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs**: **Schedule**.

Parameters

None

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Purge To Date Configurator Import Tables

The Purge To Date Configurator Import Tables concurrent program deletes data in the CZ_IMP tables, and the corresponding data in the CZ_XFR_RUN_INFOS and CZ_XFR_RUN_RESULTS control tables. The data for the number of days specified in the input parameter is retained. Periodically running this concurrent program improves import performance. See CZ Schema Maintenance , page 8-1 for more information about purging the CZ schema. See the *Oracle Configurator Performance Guide* for additional information about improving database performance.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs**: **Schedule**.

Parameters

The following table lists the parameters for the Purge To Date Configurator Import Tables Concurrent Program.

Parameter for the Purge To Date Configurator Import Tables Concurrent Program

Parameter	Description
Number of Days	This is the number of days back that you want to retain your imported data. All data imported prior to the specified number of days back is deleted.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Purge To Run ID Configurator Import Tables

The Purge To Run ID Configurator Import Tables concurrent program deletes data in the CZ_IMP tables, and the corresponding data in the CZ_XFR_RUN_INFOS, and CZ_XFR_RUN_RESULTS control tables. All subsequent data from the specified Run ID input parameter is retained. Periodically running this concurrent program improves import performance. See CZ Schema Maintenance , page 8-1 for more information about purging the CZ schema. See the *Oracle Configurator Performance Guide* for additional information about improving database performance.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs**: **Schedule**.

Parameters

Th following table lists the parameters for the Purge To Date Configurator Import Tables concurrent program.

Parameter for the Purge To Date Configurator Import Tables Concurrent Program

Parameter	Description
Run ID	This is the earliest import run ID that you want to retain. All data imported prior to the specified run ID is deleted.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Convert Publication Target Instance to Development Instance

This concurrent program converts a database that is flagged as a remote publication target instance to an instance on which you can run Oracle Configurator Developer (or use as a target when migrating Models). If, for example, a user publishes a Model from Instance A to Instance B, the latter instance becomes a publication target and users can no longer run Configurator Developer on that instance.

> **Caution:** Frequent use of this concurrent program can result in corrupt publication data or other issues. Therefore, run this program only when absolutely necessary, such as when a development instance is accidentally converted to a publication target instance.

Converting a publication target instance is a form of publication synchronization. The existing data in CZ_SERVERS, CZ_MODEL_PUBLICATIONS, and associated tables are modified to reflect the change in the network mapping of the publication. Existing publications are modified to make them consistent with the new mapping. The concurrent program updates the server definition on the source instance to indicate that the target instance has been converted.

If an error occurs while changing the instance from a remote publication target instance to a development instance, then the conversion is cancelled and the instance reverts to its prior state. This does not interrupt any Developer sessions as Developer remains disabled in the instance.

The concurrent program is run on the instance that is to be converted. Note that existing published Models are not visible in the Repository after this concurrent program is run. Once converted to a development instance, the Models can be migrated to this instance.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administratoror Configurator Administrator >Concurrent Programs:Convert Publication Target Instance to Development Instance**.

Parameters

None

Server Administration Concurrent Programs

The server administration concurrent programs are:

- Add Application to Publication Applicability List, page C-9
- Define Remote Server, page C-10
- Enable Remote Server, page C-12
- Modify Server Definition, page C-14
- View Servers, page C-13

See Database Instances, page 3-1 for information about a multi-server environment requiring use of these concurrent programs.

Add Application to Publication Applicability List

The Add Application to Publication Applicability List concurrent program adds a registered Oracle application to the CZ_EXT_APPLICATIONS table. Entries in the CZ_EXT_APPLICATIONS table are displayed in the Applications list of values parameter on the Publications page. For more information on the Applications parameter, see Applications, page 16-9.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs**: **Schedule**.

Parameters

The following table describes the parameters for the Add Application to Publication Applicability List concurrent program.

Parameter for the Add Application to Publication Applicability List Concurrent Program

Parameter	Description
Application Name	From the list of values, select the desired application. The displayed applications are registered Oracle Applications found in the FND_APPLICATION table.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Define Remote Server

The Define Remote Server concurrent program creates a new remote server definition and adds the name of the remote database instance to the:

- CZ_SERVERS table (see the Oracle Integration Repository. or the Oracle Support Web site for details)
- Database Instance publication applicability parameter list in Oracle Configurator Developer
- Target Database Instance list in Oracle Configurator Developer (used when migrating Models)
- Target Instance parameter for the Models synchronization concurrent programs
- Source Name parameter for the Setup Configurator Data Migration concurrent program
- Target Instance parameter for the Publication synchronization concurrent programs and (after running the Enable Remote Server concurrent program) the Configuration Model Publication concurrent programs

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs**:

Schedule.

Parameters

The following table describes the parameters for the Define Remote Server concurrent program

Parameters for the Define Remote Server Concurrent Program

Parameter	Description
Local Name	This is the local name for the remote instance. It is the name that: • appears in the list when creating a publication record and specifying a Target Database Instance • is in the list of values when a Target or Source Instance parameter is needed for running a concurrent program • appears in the list of values when enabling a remote server. For example, "production"
Host Name	A TCP host name for the server where the CZ schema is found. This can be an IP address or the actual name of the server. This is the actual computer. For example, ap123dbs.
DB Listener Port	A TCP port number on which this database server is listening for client connections. For example, 1523.
Instance Name	The Instance Name identifies a specific instance of the Oracle database. This is the instance name on the remote server. Also known as the SID. The Instance Name appears in the TNSNAMES.ORA file.
Oracle Applications Schema Name (FNDNAM)	The Name of Oracle Applications Schema (FNDNAM). For example, "apps".
Global Identity	When the database initialization parameter GLOBAL_NAMES is set to true, this field should be set to the name of the remote server. When GLOBAL_NAMES is true, the name of the FND Link Name must match the global name of the database you are linking to.
Description	Any notes you want to make regarding this server.

Parameter	Description
FND Link Name	The name of the remote server link to the Oracle Applications schema. For example, czvis1.world.
Import Enabled (Y/N)	Enable or disable import on the remote server. Only one remote server can be enabled for import at a time.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Enable Remote Server

Enable Remote Server concurrent program performs all the operations needed to enable a remote server for import, publishing, synchronizing and migrating data. When a remote server is enabled, the list of remote BOM Models are linked into the local instance for use by the Populate/Refresh Configuration Models concurrent program. If a remote server is going to be the source for importing data, then the **Import Enabled** parameter must be set to Y. If the remote server will be a publication target instance, then the **Import Enabled** parameter must be set to N.

For more information about importing data, see Populating the CZ Schema, page 5-1.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Enable Remote Server concurrent program

Parameters for the Enable Remote Server Concurrent Program

Parameter	Description
Server Local Name	Select from the list of values or enter the name of the server entry that you want to enable. "FOREIGN" (-1) and the local server (0) are invalid parameters.
Password	This is the password for the Oracle Applications schema (FNDNAM) on this remote server.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

View Servers

The View Servers concurrent program writes each defined server's information to the concurrent program log.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

None

Output

The log file lists each defined server's Server Name (corresponding input parameter is Local Name), Host Name, Port, Instance Name, Server Db Version, FND Name, Global Name, Notes (corresponding input parameter is Description), FND Link Name, Import Enabled. There is no indication whether the defined server has been enabled.

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Modify Server Definition

The Modify Server Definition concurrent program allows the changing of the server's previously defined input parameters. For example, if you are changing your import source from the local server to a remote server, you must run the Modify Server Definition concurrent program to change the value of the **Import Enabled** parameter for the local server in addition to defining and enabling the remote server for import.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Modify Server Definition concurrent program

Parameters for the Modify Server Definition Concurrent Program

Parameter	Description
Local Name	This is the local name for the remote instance. It is the name that: • appears in the list when creating a publication record and specifying the Database Instance applicability parameter • is in the list of values when a Target or Source Instance parameter is needed for running a concurrent program • appears in the list of values when enabling a remote server. For example, production
Host Name	A TCP host name for the server where the CZ schema is found. This can be an IP address or the actual name of the server. This is the actual computer. For example, myserver
DB Listener Port	A TCP port number on which this database server is listening for client connections.

Parameter	Description
Instance Name	The Instance Name identifies a specific instance of the Oracle database. Also known as the SID. This name appears in the TNSNAMES.ORA file.
Oracle Applications Schema Name (FNDNAM)	A Name of Oracle Applications Schema (FNDNAM).
Global Identity	When the database initialization parameter GLOBAL_NAMES is set to true, this field should be set to the name of the remote server. When GLOBAL_NAMES is true, the name of the FND Link Name must match the global name of the database you are linking to.
Description	Any notes you want to make regarding this server.
FND Link Name	The Name of the remote server link to the Oracle Applications schema. For example, czvis1.world.
Import Enabled (Y/N)	Enable or disable import on this server. Only one remote server can be enabled for import at a time.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Configuration Model Publication Concurrent Programs

The publication concurrent programs include:

- Process Pending Publications, page C-16
- Process a Single Publication, page C-17

These concurrent programs create a copy of a configuration model's structure, rules, and UI by copying the data from the development database instance to the CZ_MODEL_PUBLICATIONS table on the target **Database Instance** specified in the Oracle Configurator Developer Model Publication page.

These concurrent programs must be run in the source database. The source database is the database in which the configuration model and its publication record are defined. The target database for the publication process is specified by the publication's applicability parameters. See the *Oracle Configurator Developer User's Guide* and

Publication Applicability Parameters, page 16-8 for more information about applicability parameters.

Typically, concurrent programs are scheduled to run automatically. If for some reason you do not have these concurrent programs scheduled, or you cannot wait to publish your Model until the next scheduled request run, you can run either program manually.

The target publication database instance must be defined and enabled as a remote server unless the target server is the same as the source server. If the target server is the same as the source server, then the target server does not have to be enabled. See Server Administration Concurrent Programs, page C-9 .

Running the publication concurrent programs includes BOM Model synchronization. For details, see Checking BOM and Model Similarity, page 7-3 and Publishing a Configuration Model, page 16-10.

Process Pending Publications

The Process Pending Publications concurrent program copies all Model data (source publications) in the CZ_PB_MODEL_EXPORTS table that have a STATUS of PEN (pending) to the instance specified when creating the publication in Configurator Developer.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database where the configuration model and its publication are defined.

> **Note:** When running the Process Pending Publications concurrent program, all affected Models including referenced Models are temporarily locked while the program is running. If any affected Model is already locked by a user other than the one making the request, the concurrent program logs an error without completing the request. For details about locking objects, see the *Oracle Configurator Developer User's Guide*.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

None

Output

To see if there are any errors or warnings for the concurrent program, examine the log files. See Viewing Log Files, page B-4.

Process a Single Publication

The Process a Single Publication concurrent program copies a specific Model publication to a the instance specified when creating the publication in Configurator Developer..

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database where the configuration model and its source publication are defined.

> **Note:** When running the Process a Single Publication concurrent program, all affected Models including referenced Models are temporarily locked while the program is running. If any affected Model is already locked by a user other than the one making the request, the concurrent program logs an error without completing the request. For details about locking objects, see the *Oracle Configurator Developer User's Guide* .

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

The following table describes the parameters for the Process a Single Publication concurrent program

Parameters for the Process a Single Publication Concurrent Program

Parameter	Description
Publication	Select from the list of values or enter the publication ID of the publication you want to export to the database instance specified in the publication record. The publication ID is displayed in the Model Publication page in Oracle Configurator Developer, and is stored in the CZ_MODEL_PUBLICATIONS table.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Populate and Refresh Configuration Models Concurrent Programs

The concurrent programs for populating and refreshing configuration models are:

- Populate Configuration Models, page C-19
- Refresh a Single Configuration Model, page C-21
- Refresh All Imported Configuration Models, page C-22
- Disable/Enable Refresh of a Configuration Model, page C-23
- Import Configuration Rules, page C-23

Use the Populate/Refresh Configuration Models concurrent programs to import data into the CZ schema, including:

- Extracting BOM Model data into the correct format for transfer (Standard Import, page 5-4, only)
- Loading the data into the import tables (Standard Import, page 5-4, only)
- Populating the online CZ schema with the data from the import tables

For more information about data import, see Populating the CZ Schema, page 5-1.

> **Note:** The Populate and Refresh Configuration Models concurrent programs do not provide an automated or scheduled mechanism that clears the import tables.

Oracle does not recommend running Populate and Refresh Configuration Models Concurrent Programs, page C-18 and Import Configuration Rules, page C-23 concurrent program at the same time.

Populate Configuration Models

The Populate Configuration Models concurrent program populates the CZ schema online tables with data for creating configuration models that are based on existing BOMs or legacy data.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database into which you are importing data.

> **Note:** When running the Populate/Refresh Configuration Models concurrent program, all affected BOM Models including referenced Models are temporarily locked while the program is running. If any affected BOM Model is already locked by a user other than the one making the request, the concurrent program logs an error without completing the request. For details about locking objects, see the *Oracle Configurator Developer User's Guide*'.
>
> You cannot run simultaneous Populate/Refresh Configuration Models requests. If there is another Populate/Refresh Configuration Models running when you start the concurrent program, then your request will terminate.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

If no data is available in the list of values for the following parameters, see Populate Configuration Models Concurrent Program Error Messages, page C-20.

The following table describes the parameters for the Populate Configuration Models concurrent program

Parameters for the Populate Configuration Models Concurrent Program

Parameter	Description
Organization Code	Required. Select from the list of values or enter the BOM Models' Inventory organization that you want to import the BOM Models from.
Model Inventory Item From	Select the first Model Inventory Item in the range of BOM Models you want to import. All Model Inventory Items between and including the first and last specified in this and the next field, are included in the data import. The range can include multiple types of Model Inventory Items. For example, from ATO800 to PTO500 is a valid range.
Model Inventory Item To	Select the last Model Inventory Item in the range of items for which you want to import data. If you want to import a single model, enter the same Model Inventory Item that you entered for the Model Inventory Item From parameter.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Populate Configuration Models Concurrent Program Error Messages

On certain error conditions there is no data in the extraction views and the list of values for a new import does not have any data. In these cases, the list of values for the Populate Configuration Models concurrent program displays a `'No entries found for List of Values'` message. Possible reasons for the missing data include:

- The server's Enabled for Import parameter has not been set to Y
- The Enable Remote Server concurrent program did not complete successfully
- The database link is down
- The remote database is down
- The extraction views are invalid

If the database link is down, the following message appears:

Example

```
'error 2019: connection description for remote database not found'
```

(The SQL statement that is currently running follows this message.)

Action:

1. The Oracle Configurator Administrator must run the Modify Server Definition, page C-14 concurrent program and Enable Remote Server, page C-12 for import (if one is not already selected). See Parameters for the Modify Server Definition Concurrent Program, page C-14 and Parameters for the Enable Remote Server Concurrent Program, page C-13.

2. Run Enable Remote Server, page C-12 if the enabled server is not LOCAL. See Parameters for the Enable Remote Server Concurrent Program, page C-13. Rerunning this concurrent program recreates the extraction views.

Refresh a Single Configuration Model

The Refresh a Single Configuration Model concurrent program updates the imported BOM Model data in the CZ schema when information in Oracle Applications **Bills of Material** and Inventory has changed.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database in which you are refreshing data.

> **Note:** When running the Refresh a Single Configuration Model concurrent program, all affected BOM Models including referenced Models are temporarily locked while the program is running. If any affected BOM Model is already locked by a user other than the one making the request, the concurrent program logs an error without completing the request. For details about locking objects, see the *Oracle Configurator Developer User's Guide*.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

Parameters for the Refresh a Single Configuration Model and Disable/Enable Refresh Concurrent Programs, page C-22 lists the parameters used for the Refresh a Single Configuration Model concurrent programs.

The following table describes the parameters for the Refresh a Single Configuration Model and the Disable/Enable Refresh concurrent programs.

Parameters for the Refresh a Single Configuration Model and Disable/Enable Refresh Concurrent Programs

Parameter	Description
Folder	Enter the name of the Configurator Developer Repository Folder in which the configuration model resides, or select a Folder from the list of values.
Configuration Model ID	Select a Model from the list of values.

Output

To see if there are any errors or warnings for the concurrent program, examine the log files. See Viewing Log Files, page B-4.

Refresh All Imported Configuration Models

The Refresh All Imported Configuration Models concurrent program updates all of your imported BOM Model data.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database in which you are refreshing data.

> **Note:** When running the Refresh All Imported Configuration Models concurrent program, all affected BOM Models including referenced Models are temporarily locked while the program is running. If any affected BOM Model is already locked by a user other than the one making the request, the concurrent program logs an error without completing the request. For details about locking objects, see the *Oracle Configurator Developer User's Guide*.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

None

Disable/Enable Refresh of a Configuration Model

The Disable/Enable Refresh of a Configuration Model concurrent program prevents (disables) or allows (enables) either of the Refresh Configuration Model concurrent programs to update a specific configuration model. You may want to prevent a configuration model from being updated if you are currently designing its configuration rules in Configurator Developer. This concurrent program is run in the database instance where the configuration model resides.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database containing the configuration model whose refresh is being controlled.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

The following table lists the parameters for the Disable/Enable Refresh of a Configuration Model concurrent program.

Parameters for the Disable/Enable Refresh Concurrent Program

Parameter	Description
Folder	Enter the name of the Configurator Developer Repository Folder in which the configuration model resides, or select a Folder from the list of values.
Configuration Model ID	Select a Model from the list of values.
Refresh Enabled (Y/N)	The response of Yes or No indicates whether the specified Model is refreshed when the Refresh concurrent programs are run.

Import Configuration Rules

The Import Configuration Rules concurrent program imports rules that are written in Constraint Definition Language format into the CZ schema. For more information, see Rule Import, page 5-21.

Note: You cannot run simultaneous Import Configuration Rules requests. If there is another Import Configuration Rules request running, then your rule import request will terminate.

If the rule's Model is locked, then an appropriate message is returned and the configuration rules are not imported into the CZ schema.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

Thefollowing table lists the parameters for the Import Rules concurrent program.

Parameter for the Import Configuration Rules Concurrent Program

Parameter	Description
Run ID	This is an optional import session parameter. Run ID identifies a set of source data that is converted into rules after the data is imported into the CZ schema. If this parameter is null, then the records in CZ_IMP_RULES with RUN_ID, REC_STATUS, and DISPOSITION that are NULL are imported into the CZ schema and will have a generated RUN_ID. If Run ID is not null, then all records in CZ_IMP_RULES with the given RUN_ID are processed and refreshed in the CZ schema. If the Run ID is an invalid Run ID, then the following message is returned: `'No data found in the CZ_IMP_RULES table with RUN_ID = &RUNID'`.

Note: If you want to refresh a set of rules that have the same Run ID, you must first manually set CZ_IMP_RULES.REC_STATUS, CZ_IMP_RULES.DISPOSITION, CZ_IMP_LOCALIZED_TEXTS.REC_STATUS, and CZ_IMP_LOCALIZED_TEXTS.DISPOSITION to NULL for those records that have the desired Run ID. You then run the Import

Configuration Rules concurrent program with the Run ID. Note that any changes made to the rule in Configurator Developer will be overridden with the newly imported rule.

Output

Rules are validated for CDL structure and rule participants. If an imported rule has a parsing error, the parsing error is written in both the concurrent program log file and CZ_IMP_RULES.MESSAGE. The REC_STATUS for CZ_IMP_RULES and CZ_IMP_LOCALIZED_TEXTS is ERR, and CZ_IMP_LOCALIZED_TEXTS.DISPOSITION is R. Rules imported into the CZ schema can be edited either in Configurator Developer or the source environment.

The rule import run data is logged in the CZ_XFR_RUN_INFOS table as well as the CZ_XFR_RUN_RESULTS table. For more information about these tables and the tables used during rule import, see the Oracle Integration Repository or the Oracle Support Web site.

Any concurrent program errors or warnings are in the FND log file. See Viewing Log Files, page B-4.

Model Synchronization Concurrent Programs

The model synchronization concurrent programs include:

- Check Model/Bill Similarity, page C-25
- Check All Models/Bills Similarity, page C-27
- Synchronize All Models, page C-27

Check Model/Bill Similarity, page C-25 and Check All Models/Bills Similarity, page C-27 compare the imported model and the BOM Model to see if they are similar enough to synchronize. If key validation fields are not equal, then the requests generate a Model/Bill Similarity Check Report, page C-28 listing the fields with discrepancies. The user must resolve the discrepancies before synchronizing the models. This is an iterative process. Once the validation fields are corrected and the report no longer returns discrepancies, the Synchronize All Models, page C-27 can be run.

See Synchronizing Data, page 7-1for more information.

Check Model/Bill Similarity

The Check Model/Bill Similarity concurrent program compares a single configuration model with the BOM Model on which it is based, and produces a Model/Bill Similarity Check Report, page C-28 of discrepancies, if any. See Criteria for BOM Model Similarity, page 7-3 for information about the validation criteria used by this concurrent

program.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database containing the configuration model that must be checked.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

The following table describes the parameters for the Check Model/Bill Summary concurrent program

Parameters for the Check Model/Bill Similarity Concurrent Program

Parameter	Description
Target Instance	A list of available instances, as defined by the Define Remote Server, page C-10 concurrent program. Select the instance that contains the source BOM Model with which the configuration model must be synchronized.
Folder	A list of folders (Configurator Developer Repository Folders) on the specified Target instance. Select the Folder that contains the Model to be checked against the BOM Model in the Target Instance.
List of Models	A list of all Models in the specified Folder on the specified Target instance. Select a Model from the list of values.

Output

A report is generated with the results. See Model/Bill Similarity Check Report, page C-28.

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Check All Models/Bills Similarity

The Check All Model/Bill Similarity concurrent program compares all configuration modes in the local database instance with the BOM Models on which they are based, and produces a Model/Bill Similarity Check Report, page C-28 of discrepancies, if any. See Criteria for BOM Model Similarity, page 7-3 for information about the validation criteria used by this concurrent program.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database containing the configuration models that need to be checked.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

The following table describes the parameters for the Check All Models/Bills Similarity concurrent program

Check All Models/Bills Similarity Parameters

Parameter	Description
Target Instance	A list of available instances, as defined by the Define Remote Server, page C-10 concurrent program. Select the instance that contains the source BOM Model with which the configuration model must be synchronized.

Output

A report is generated with the results. See Model/Bill Similarity Check Report, page C-28.

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Synchronize All Models

The Synchronize All Models concurrent program modifies the configuration models on

the local database instance to match the corresponding BOM Models in the Bills of Material schema that is to serve as the new import server or publication target. All imported models in the CZ schema of the current instance are synchronized with the corresponding structures of the bills on a target instance.

The Synchronize All Models concurrent program is run after all errors and discrepancies in the report generated by the Check All Models/Bills Similarity, page C-27 concurrent program have been corrected (see Model/Bill Similarity Check Report, page C-28).

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database containing the configuration model that must be synchronized.

> **Warning:** Oracle Configurator Developers must not modify Models when the Synchronize All Models concurrent program is running.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

None

Output

A report is generated with the results. See Model/Bill Similarity Check Report, page C-28.

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Model/Bill Similarity Check Report

The Model/Bill Similarity Check Report is generated every time you run the Check Model/Bill Similarity, page C-25, Check All Models/Bills Similarity, page C-27 and Synchronize All Models, page C-27 concurrent programs. The report is displayed in a standard report log file generated by concurrent programs. For detailed information on concurrent processing reporting options, see the *Oracle Application's User's Guide*. For a list of validation criteria used to generate the report, see Criteria for BOM Model Similarity, page 7-3.

The Model/Bill Similarity Check Report contains a comprehensive message describing

the list of problems that were encountered. The list starts with a message providing the version of the package and the run time. For example, the following message occurs when the BOM Model does not exist on the target server:

Example

```
BOM Synchronization, version 115.15, started 2001/10/23/14:05:16,
session run ID: 12017
There is no root bill for configuration model Name of the Model, unable
to verify the model."
```

The following message occurs when there is a discrepancy with an Inventory Item.

Example

```
BOM Synchronization, version 115.15, started 2001/10/29/14:05:16,
session run ID: 12018
'PTO_OC1' with parent 'BOM_SYNCH' in configuration model 'BOM_SYNCH'
cannot be matched with any inventory item.
```

Execute Populators in Model

Use this concurrent program to run all Populators defined for a specific Model. This concurrent program allows you to automate the process of updating a Models with data that has changed in the Item Master For information about Populators, see the *Oracle Configurator Developer User's Guide*.

Do not confuse the Execute Populators in Model concurrent program with the Populate Configuration Models concurrent program. You run the Populate Configuration Models program to import data from Oracle Bills of Material. The Execute Populators in Model concurrent program updates Model structure that was created in Configurator Developer from data in the CZ schema's Item Master.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the database containing the Models you want to repopulate.

Responsibility

Oracle Configurator Administrator or Oracle Configurator Developer

Navigation

Navigator window >**Oracle Configurator Administrator** or **Oracle Configurator Developer** >**Concurrent Programs:Schedule**.

Parameters

The following table describes the parameters for the Execute Populators in Model concurrent program

Parameters for the Execute Populators in Model Concurrent Program

Parameter	Description
Folder	A list of folders (Configurator Developer Repository Folders) on the current instance. Select the Folder that contains the Model in which you want Populators to be implemented.
Configuration Model ID	Select a Model from the list of values. Configuration Model ID is the same ID as the DEVL_PROJECT_ID.

Output

To see if there are any errors or warnings for the concurrent program, examine the log files. See Viewing Log Files, page B-4.

Migration Concurrent Programs

Migration concurrent programs move data from a source CZ schema to an empty target CZ schema, or copy Models from one database instance to another development database instance.

The migration concurrent programs that move data from a source CZ schema to an empty target CZ schema are:

- Setup Configurator Data Migration, page C-30
- Migrate Configurator Data, page C-32

See Migrating Data from A CZ Schema, page 6-2 for prerequisites before running the migration concurrent programs.

The source database server must be defined and enabled as the remote server (see Server Administration Concurrent Programs, page C-9.

The migration concurrent program that copies Models from one database instance to another development database instance is:

- Migrate Models, page C-38

Setup Configurator Data Migration

The Setup Configuration Data Migration concurrent program identifies the source database instance of a data migration.

Use the procedure described in Running Configurator Concurrent Programs, page B-2

to run this concurrent program in the target database into which you are migrating data.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Setup Configurator Migration concurrent program

Parameters for the Setup Configurator Data Migration Concurrent Program

Parameter	Description
Source	Enter the name of the source database instance containing the data to be migrated, or select a source from the list of values defined by the Define Remote Server, page C-10 concurrent program.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file (see Viewing Log Files Program, page B-4).

In the Request concurrent program, view the log file to verify that no issues were found during the migration setup. Possible issues are:

- Specified instance name does not have an associated database link
- Associated database link is not functional
- Database error occurred during the population of the control tables
- Schema versions for the source and target databases are not the same
- Difference in table structure

If any issues are reported, correct them and run Setup Configuration Data Migration again.

Migrate Configurator Data

The Migrate Configurator Data concurrent program migrates the data from the source database instance to the target database instance. See Migrating Data, page 6-1 for migration requirements.

Use the procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program in the empty target database into which you want to migrate data.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Migrate Configurator Data concurrent program

Parameters for the Migrate Configurator Data Concurrent Program

Parameter	Description
Proceed when database not empty?	Enter Yes or No to this prompt. The migration should only be run against an empty target database. However, if for some reason the original migration does not complete successfully (for example a roll back segments problem), then the migration must be rerun after the roll back segments problem has been resolved. If the migration is repeated after such a correction, then the `Proceed when database not empty?` prompt can be answered Yes

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Migrate Functional Companions

The Functional Companion migration concurrent programs are:

- Migrate All Functional Companions, page C-33
- Migrate Functional Companions for a Single Model, page C-34

These concurrent programs transform existing Functional Companion association data in the database to the new form of association data used for Configurator Extensions.

After you upgrade to the release of Oracle Configurator described in this document, you may need to migrate Functional Companions that were created with previous releases.

See the *Oracle Configurator Installation Guide* for background information.

Migrate All Functional Companions

The Migrate All Functional Companions concurrent program creates Configurator Extension associations for all Functional Companions in the database.

> **Note:** You must perform some setup tasks before and after running this concurrent program. See the *Oracle Configurator Installation Guide*.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

None

Output

If the migration finishes without errors, then Configurator Extension Rules (association data) are created for all Functional Companions in the database.

> **Warning:** After you successfully migrate Functional Companions to Configurator Extensions, all existing Functional Companion data is logically deleted from the database.

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4. If errors occur while processing a Model, the migration for that Model stops, and all transactions are rolled back. Processing continues for other Models in the database.

Migrate Functional Companions for a Single Model

The Migrate Functional Companions for a Single Model concurrent program creates Configurator Extension associations for the Functional Companions associated with a specified Model.

> **Note:** You must perform some setup tasks before and after running this concurrent program. See the *Oracle Configurator Installation Guide*.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Migrate Functional Companions for a Single Model concurrent program.

Parameters for the Migrate Functional Companions for a Single Model Concurrent Program

Parameter	Description
Model ID	This is the Model that contains Functional Companions to be migrated. A list of all Models is available to select from, including those that do not contain Functional Companions at all and those that do not contain Functional Companions but whose child Models contain Functional Companions. If you choose a Model that does not contain Functional Companions then the migration still runs, but the migration log shows that the Model contained no Functional Companions. To migrate Functional Companions that are contained in any child Models, you must choose the option for deep migration.
Migrate Child Model's FC	This is a Yes/No flag indicating whether you want the concurrent program to perform a deep migration. A Yes response means that all of the Functional Companions associated with the selected Model and its child Models will be migrated.

Output

If the migration finishes without errors, then Configurator Extension Rules (association data) are created for all Functional Companions associated with the selected Model.

> **Warning:** After you successfully migrate Functional Companions for a Model to Configurator Extensions, the existing Functional Companion data for the Model is logically deleted from the database.

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4. If no errors occur in the migration, then all transactions related to the specified Model are committed. If errors occur while processing the Model, the migration process stops and all transactions are rolled back.

Model Management

The Model Management concurrent programs are:

- Add Model Node Names to Configurations by Model Items, page C-35
- Add Model Node Names to Configurations by Model Product Key, page C-37
- Migrate Models, page C-38

Add Model Node Names to Configurations by Model Items

Before the Model migration functionality was introduced, the persistent node ID was a consistent set of references to the same Model node. When Models are migrated to other development database instances, the persistent node ID becomes a one-to-many relationship, and can no longer be relied upon when identifying a saved configuration.

The Add Model Node Names to Configurations by Model Items concurrent program adds Model node names to saved configurations of a Model based on the Item or range of Items that you specify. It enables Oracle Configurator to uniquely identify Model nodes using their *names* rather than persistent_node_id values when restoring saved configurations. This is required to successfully restore configurations that were saved against migrated Models.

Alternatively, you can use the Add Model Node Names to Configurations by Model Product Key, page C-37 concurrent program to update saved configurations.

You run this concurrent program in the database instance where the migrated Models are located. To run this concurrent program, use the procedure described in the Running Configurator Concurrent Programs, page B-2.

If the Model for a deleted publication exists, then the configurations for the deleted publication are upgraded using the configuration's source Model. The example Saved Configurations and a Copied Model, page C-36 shows what can prevent Oracle

Configurator from identifying all of the inputs in a saved configuration, resulting in a validation failure message.

Saved Configurations and a Copied Model

There are two Models, ModelA and its copy, ModelA'. In Oracle Configurator Developer, a new node called Option1 is added to ModelA. ModelA is published and a configuration is saved against ModelA. A new node Option1 is added to ModelA', and ModelA' is published.

When restoring an order saved against ModelA, a validation failure message appears because Oracle Configurator is unable to match Option1 that was added to ModelA' with Option1 in ModelA.

Responsibility

Oracle Configurator Administrator or Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator > Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Add Model Node Names to Configurations by Model Items concurrent program.

The parameters you specify determine which published Models will be used to update the configurations. The values of the profile options CZ: Publication Usage and CZ: Publication Lookup Mode are also used to further refine the search for published Models. Oracle Configurator profile options are described in the *Oracle Configurator Installation Guide*.

Parameters for the Add Model Node Names to Configurations by Model Items

Parameter	Description
Organization	The Inventory Organization of the Item(s) you want to update
Model Inventory Item From	The beginning of a range of Models to be processed (To update saved configurations of a single Model, enter the same value for this parameter and the 'Model Inventory Item To' parameter.)
Model Inventory Item To	The end of a range of Models to be processed

Parameter	Description
Application	Enter a valid host application to update only configurations published for that application.
Configuration Begin Date	The date of the oldest configuration to be updated (use the format DD-MON-YYYY)
Configuration End Date	The date of the newest configuration to be updated (use the format DD-MON-YYYY) The default date is the system date.

Output

To see if there are any errors or warnings for the concurrent program, examine the CZ_DB_LOGS file. For details, see Viewing Log Files, page B-4.

Add Model Node Names to Configurations by Model Product Key

This concurrent program is similar to Add Model Node Names to Configurations by Model Items, page C-35, but this program adds Model node names to saved configurations based on the *Product Key* that you specify, rather than an Item or range of Items.

You run this concurrent program in the database instance where the migrated Models are located. To run this program, use the procedure described in Running Configurator Concurrent Programs, page B-2.

Responsibility

Oracle Configurator Administrator or Configurator Administrator

Navigation

Navigator window > **Oracle Configurator Administrator > Concurrent Programs: Schedule**.

Parameters

The following table lists the parameters for the Add Model Node Names to Configurations by Model Product Key concurrent program.

The parameters you specify determine which published Models will be used to update the configurations. The values of the profile options CZ: Publication Usage and CZ: Publication Lookup Mode are also used to further refine the search for published Models. Oracle Configurator profile options are described in the *Oracle Configurator*

Installation Guide.

Parameters for the Add Model Node Names to Configurations by Model Product Key

Parameter	Description
Product key	The Item's Inventory Organization and Item number. For example: `204:5717`.
Application	Enter a valid host application to update only configurations published for that application.
Configuration Begin Date	The date of the oldest configuration to be updated (use the format DD-MON-YYYY)
Configuration End Date	The date of the newest configuration to be updated (use the format DD-MON-YYYY) The default date is the system date.

Output

To see if there are any errors or warnings for the concurrent program, examine the log files. See Viewing Log Files, page B-4.

Migrate Models

This concurrent program migrates (copies) a configuration model its to another database instance. For details about what data is migrated, see Migrating Models, page 6-3.

For information about how data is synchronized when Models are migrated, see Synchronizing Migrated Model Data, page 6-8.

Before running this concurrent program, you must:

- Define and enable the target instance as a remote server.

  For details, see Define Remote Server, page C-10 and Enable Remote Server, page C-12

  > **Note:** Models cannot be migrated to a remote publication target instance. For details, see Convert Publication Target Instance to Development Instance, page C-8.

- Identify the Models you want to migrate in Oracle Configurator Developer.

For details, see the *Oracle Configurator Developer User's Guide.*

When you identify the Models to be migrated, Configurator Developer generates a Migration Group ID. You specify this value for the Migration Group parameter when running the Migrate Models concurrent program.

While the Migrate Models concurrent program is running, the specified root Model and all of its referenced Models are locked in Configurator Developer. If the program cannot lock any of the Models (for example, because they are locked by another user), the program fails and displays an error. Model locking is explained in the *Oracle Configurator Developer User's Guide.*

To migrate Models, the target and source instances must be at the same Oracle Configurator patch level, and the same sets of languages must be installed on each instance.

When the Migrate Models concurrent program completes successfully, review the log file to review the results of the migration. For details, see Synchronizing Migrated Model Data, page 6-8.

> **Important:** After running the Migrate Models program, review Restoring Saved Configurations of Migrated Models, page 6-7 for important information.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Migrate Models concurrent program.

Parameters for the Migrate Models Concurrent Program

Parameter	Description
Migration Group	The Migration Group ID that is generated when you identify the Model(s) to be migrated in Oracle Configurator Developer. This number identifies the Models to be migrated from the source instance and the destination folder on the target instance.

Output

To see if there are any errors or warnings for the concurrent program, examine the log files. See Viewing Log Files, page B-4.

Publication Synchronization Concurrent Programs

The publication synchronization concurrent programs are:

- Synchronize Cloned Target Data, page C-40
- Synchronize Cloned Source Data, page C-41

These concurrent programs resolve data inconsistencies that result when a source or target database instance is cloned or migrated from a different database instance.

Publication synchronization updates publication record pointers to servers, checks overlaps of applicability parameters and item definitions, and realigns relationships between publication records that became invalid.

Before running these concurrent programs, the target database instance must be defined and enabled to establish the database link. See Server Administration Concurrent Programs, page C-9 for information about defining and enabling a remote server.

Synchronize Cloned Target Data

The Synchronize Cloned Target Data ensures that publication data on the cloned publication target database instance matches that on the publication source database instance. For example, you have published models and are working with two database instances: a publication source and a publication target. You then clone the publication target. The publication data on the cloned publication target does not recognize the publication source until you run the Synchronize Cloned Target Data. For more information see Synchronizing Publication Data after a Database Instance is Cloned, page 7-7.

If the publication records on the target exist on the source database instance, then the SERVER_ID of the target publication is updated and a new publication record is created on the source database instance with updated references.

Note: Do not:

- Publish or republish Models when the synchronization concurrent programs are running
- Synchronize publications when publishing or republishing Models

The Synchronize Cloned Target Data concurrent program must be run in the database instance that serves as the publication source for the cloned publication target. Use the

procedure described in Running Configurator Concurrent Programs, page B-2 to run this concurrent program.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for Synchronize Cloned Target Data.

Synchronize Cloned Target Data

Parameter	Description
Target database instance	Enter the name of the cloned publication target database instance, or select a cloned publication target from the list of values defined by the Define Remote Server, page C-10 concurrent program.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

If the Model and UI in the target database instance publication record do not match the Model and UI in the source database instance publication record from which the Synchronize Cloned Target Data concurrent program is running, then the program logs an error, and the concurrent program terminates. The Model Publication page in Oracle Configurator Developer displays Error in the **Status** column (see the *Oracle Configurator Developer User's Guide*).

Synchronize Cloned Source Data

The Synchronize Cloned Source Data ensures that publication data on the publication target database instance points to the cloned publication source database instance. For example, you have published models and are working with two database instances: a publication source and a publication target. You then clone the publication source. The publication data on the publication target does not recognize the publication source until you run the Synchronize Cloned Source Data. For more information see Example of Synchronizing Publication Data on a Cloned Source, page 7-12.

Before running the concurrent program, the cloned source database instance must be defined and enabled to establish the database link. See Define Remote Server, page C-10 and Enable Remote Server, page C-12. This concurrent program is run from the cloned source database instance.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters for the Synchronize Cloned Source Data concurrent program

Synchronize Cloned Source Data

Parameter	Description
Decommission Original Source? (Yes/No)	If the original source server is decommissioned, then the CZ_SERVERS.SOURCE_SERVER_FLAG on the target is updated to no longer point to the original source server. If the original source server is not decommissioned, then the publication entries are logically deleted from the tables on the cloned source server to avoid conflicts with the original publication source.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Select Tables to be Imported

You may want to specify only a group of tables from which extracted data is loaded into the import tables. The CZ_XFR_TABLES.DISABLED field determines if a specific table is enabled or disabled for import.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

All parameters for this concurrent program are required.

The following table describes the parameters Import Data into Specific Tables concurrent program

Import Data into Specific Tables

Parameter	Description
Name	This is a list of concurrent programs. Select the Select Tables To Be Imported concurrent program from the list.
Destination Table Name	This is a list of tables in the CZ schema for which import is enabled or disabled. The table names display with a description of Import, Extract, Generic, or Populators. Be sure to select the table name with the appropriate description.
Import Group	From the list of values, select the name of the phase or group in which import is to be enabled or disabled for the specified table: Export, Import or Generic
Enable (Y or N)	From the list of values, select N to disable or Y to enable the specified table for the specified import phase.

Importing Data into a Specific Table

The following is an example that enables a table for importing data.

```
Destination Table Name: CZ_ITEM_MASTERS
Import Group: Import
Enable:Y
```

Action

After specifying the parameters click **OK**. In the Submit Request window, click **Submit**.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Viewing Log Files, page B-4.

Show Tables to be Imported

You can display the tables that are currently enabled for import.

Responsibility

Oracle Configurator Administrator

Navigation

Navigator window >**Oracle Configurator Administrator** >**Concurrent Programs: Schedule**.

Parameters

The following table describes the parameters Show Tables to be imported requested task.

Show Tables to be Imported

Parameter	Description
Table Name	Enter the table in the CZ schema that you are querying the import disability.
Import Group:	Enter either Extract, Generic, or Import for which you want to display the Import Enable setting.

Show Tables to be Imported

The following example displays the current Disable setting for the CZ_XFR_TABLES.

```
Table Name: CZ_ITEM_MASTERS
Phase Name: Import
```

Action

After specifying the parameters click **OK**. In the Submit Request window, click **Submit**.

Output

To see if there are any errors or warnings for the concurrent program, examine the log files. See Viewing Log Files, page B-4. The return for Show Tables to be Imported, page C-44:

Return from the Show Tables to be Imported Concurrent Program

```
DST_TABLE = CZ_ITEM_MASTERSXFR_GROUP = IMPORTDISABLED_FLAG = 0
```

D

CZ Subschemas

This appendix lists the CZ tables that make up each of the subschemas in the CZ schema. For table details, see the Oracle Integration Repository.

This appendix covers the following topics:

- Oracle Configurator Subschemas

Oracle Configurator Subschemas

The following sections list the tables in each subschema. For detailed information about these and other tables, see the Oracle Integration Repository.

ADMN Administrative Tables

These tables are uses for customizable site parameters and auditing information.

- CZ_DB_LOGS
- CZ_DB_SETTINGS
- CZ_DB_SIZES

CNFG Configuration Tables

These tables hold configuration information.

- CZ_CONFIG_ATTRIBUTES
- CZ_CONFIG_CONTENTS_V
- CZ_CONFIG_DETAILS_V
- CZ_CONFIG_EXT_ATTRIBUTES

- CZ_CONFIG_HDRS
- CZ_CONFIG_HDRS_V
- CZ_CONFIG_INPUTS
- CZ_CONFIG_ITEMS
- CZ_CONFIG_ITEMS_V
- CZ_CONFIG_MESSAGES
- CZ_CONFIG_MESSAGES_V
- CZ_CONFIG_USAGES

ITEM Item-Master Tables

Thefollowing tables store information about Items that are used to build a Model:

- CZ_IMP_ITEM_MASTER
- CZ_IMP_ITEM_PROPERTY_VALUE
- CZ_IMP_ITEM_TYPE
- CZ_IMP_ITEM_TYPE_PROPERTY
- CZ_IMP_PROPERTY
- CZ_ITEM_MASTERS
- CZ_ITEM_PROPERTY_VALUES
- CZ_ITEM_TYPES
- CZ_ITEM_TYPE_PROPERTIES
- CZ_PROPERTIES

LCE Logic for Configuration Tables

These tables store the generated logic for a Model.

- CZ_LCE_CLOBS
- CZ_LCE_HEADERS

- CZ_LCE_LINES
- CZ_LCE_LOAD_SPECS
- CZ_LCE_OPERANDS
- CZ_LCE_TEXTS

PB Publication Tables

These tables store information that is used when publishing a Model.

- CZ_EFFECTIVITY_SETS
- CZ_EXT_APPLICATIONS
- CZ_EXT_APPLICATIONS_V
- CZ_MODEL_PUBLICATIONS
- CZ_MODEL_USAGES
- CZ_MODEL_USAGES_TL
- CZ_PB_CLIENT_APPS
- CZ_PB_LANGUAGES
- CZ_PB_MODEL_EXPORTS
- CZ_PB_TEMP_IDS
- CZ_PUBLICATION_USAGES
- CZ_SRC_MODEL_PUBLICATIONS_V

PRC Pricing Tables

These tables are used to pass pricing and configuration information to a PL/SQL callback procedure that is used for calculating **ATP** (availability-to-promise).

- CZ_ATP_REQUESTS
- CZ_PRICING_STRUCTURES

PROJ Project Structure Tables

These tables are used to store project information in Oracle Configurator Developer for building configuration models.

- CZ_COMMON_CHILDNDPROPS_V
- CZ_CONVERSION_RELS_V
- CZ_DATA_TYPES_V
- CZ_DEVL_PROJECTS
- CZ_EXPLMODEL_NODES_V
- CZ_EXPLNODES_WITHIMAGES_V
- CZ_FUNC_COMP_SPECS
- CZ_IMP_DEVL_PROJECT
- CZ_IMP_MODEL_REF_EXPLS
- CZ_IMP_PS_NODES
- CZ_MODELS_V
- CZ_MODEL_ARCHIVES_V
- CZ_MODEL_BOMREF_COUNTS_V
- CZ_MODEL_REF_EXPLS
- CZ_NODE_CAPTION_PROPERTIES_V
- CZ_NODE_JAVA_PROPERTIES_V
- CZ_NODE_NO_PROPERTIES_V
- CZ_NODE_RULE_PROPERTIES_V
- CZ_NODE_USER_PROPERTIES_V
- CZ_POPULATORS
- CZ_PSNODE_REFRULE_IMAGES_V
- CZ_PSNODE_REFUI_IMAGES_V

- CZ_PSNODE_RULE_REFS_V
- CZ_PSNODE_WITH_UIREFS_V
- CZ_PS_NODES
- CZ_PS_PROP_VALS
- CZ_SRC_DEVL_PROJECTS_V
- CZ_SYSTEM_PROPERTIES_V
- CZ_SYSTEM_PROPERTY_RELS_V
- CZ_TEMPLATE_DEFS_V
- CZ_TERMINATE_MSGS
- CZ_TERMINATE_MSGS_V
- CZ_TGT_MODEL_PUBLICATIONS_V

RP Repository Tables

These tables are used for actions performed in the Repository as well as references to Models, Effectivity Sets, and Usages.

- CZ_ACCESS_SUMMARY_LKV
- CZ_ACTIONDISPLAYUPDT_LKV
- CZ_ACTIONMODELINTER_LKV
- CZ_ACTIONNAV_LKV
- CZ_ACTIONRULENODES_LKV
- CZ_ACTIONSESSIONCTRL_LKV
- CZ_ACTIONSONMODELNODES_LKV
- CZ_ACTIONSONREPOSITORYN_LKV
- CZ_ACTIONTYPEGROUP_LKV
- CZ_AMPM_LKV
- CZ_ANYALLTRUE_LKV

- CZ_ARCHIVES
- CZ_ARCHIVES_PICKER_V
- CZ_ARCHIVE_REFS
- CZ_ASSOCIATEDMODELNODE_LKV
- CZ_BASIC_LAYOUT_REGION_LKV
- CZ_CAPCONFIGSYSPROP_LKV
- CZ_CAPMSGSYSPROP_LKV
- CZ_CAPNODESYSPROP_LKV
- CZ_CFGEXT_ARGS_SPEC_TYPE_LKV
- CZ_CFGEXT_EVENT_SCOPE_LKV
- CZ_CFGEXT_INST_SCOPE_LKV
- CZ_CFGEXT_SYSTEM_PARAMS_LKV
- CZ_CFG_SAVEASBEHAVIOR_LKV
- CZ_CFG_SEARCHCRITERIA_LKV
- CZ_COMPAT_TEMPL_SIGS_V
- CZ_COPYDESTINATION_LKV
- CZ_COPYSOURCE_LKV
- CZ_CREATEOPTIONPSNODETY_LKV
- CZ_CREATEPSNODEPSNODETY_LKV
- CZ_CREATEREPOSITORYOBJE_LKV
- CZ_CREATERULEOBJECT_LKV
- CZ_DATATYPE_LKV
- CZ_DETAILEDRULETYPES_LKV
- CZ_DETLSELECTIONSTATE_LKV

- CZ_EFFECTIVITYMETHODS_LKV
- CZ_EFFECTIVITYTYPE_LKV
- CZ_EFFSETS_PICKER_V
- CZ_EVENTTYPES_LKV
- CZ_EXNEXPRTYPE_LKV
- CZ_FEATURETYPE_LKV
- CZ_HORIZONTALALIGNMENT_LKV
- CZ_HOURS_LKV
- CZ_ICONLOOKUP_LKV
- CZ_IMAGELOOKUPS_V
- CZ_ITEMMASTEROPS_LKV
- CZ_ITEMTYPEOPERATOR_LKV
- CZ_ITEMTYPE_LKV
- CZ_JAVASYSPROPVALS_LKV
- CZ_LAYOUTREGIONS_LKV
- CZ_LAYOUT_UI_STYLE_LKV
- CZ_LISTLAYOUTREGIONS_LKV
- CZ_LOCK_HISTORY
- CZ_LOGICRULE_LKV
- CZ_LOOKUP_VALUES
- CZ_LOOKUP_VALUES_VL
- CZ_MDLNODE_CPDST_LKV
- CZ_MDLNODE_CPSRC_LKV
- CZ_MENUITEMTYPES_LKV

- CZ_MENUTYPES_LKV
- CZ_MINUTES_LKV
- CZ_MODEL_REFERENCES_PICKER_V
- CZ_MSGLISTLAYOUTREGIONS_LKV
- CZ_NODEINSTANTIABILITY_LKV
- CZ_NODELISTLAYOUTREGIONS_LKV
- CZ_NODELIST_LAYOUT_REGION_LKV
- CZ_OTHERCONTENT_LKV
- CZ_PROPERTY_PICKER_V
- CZ_PSNODERELATION_LKV
- CZ_PSNODETYPE_LKV
- CZ_PUBLICATIONMODE_LKV
- CZ_RECALCULATEPRICES_LKV
- CZ_REPOSCREATEOPS_LKV
- CZ_REPOSITORYCOPYDESTIN_LKV
- CZ_REPOSITORYCOPYMODELO_LKV
- CZ_REPOSITORY_MAIN_HGRID_V
- CZ_REPOS_TREE_V
- CZ_RPOBJECTTYPES_LKV
- CZ_RP_BOM_MODELS_V
- CZ_RP_DIRECTORY_V
- CZ_RP_EFF_DIRECTORY_V
- CZ_RP_ENTRIES
- CZ_RP_PRJ_DIRECTORY_V

- CZ_RP_USG_DIRECTORY_V
- CZ_RTCONDCOMPAR_LKV
- CZ_RTCONDOBJSETTINGS_LKV
- CZ_RULERADIOGROUP_LKV
- CZ_RULETYPECODES_LKV
- CZ_RULEUNSATMESSAGECHOI_LKV
- CZ_RULEVIOLATIONMESSAGE_LKV
- CZ_SERVERS
- CZ_SIMPLECONTROLS_LKV
- CZ_SORTORDER_LKV
- CZ_SOURCEENTITYTYPES_LKV
- CZ_SUBTYPEBOMMODEL_LKV
- CZ_SUBTYPEBOMOPTIONCLAS_LKV
- CZ_SUBTYPEBOMSTDITEM_LKV
- CZ_SUBTYPECOMPONENT_LKV
- CZ_SUBTYPEFEATURE_LKV
- CZ_SUBTYPEFEATUREGROUP_LKV
- CZ_SUBTYPEOPTION_LKV
- CZ_SUBTYPEPRODUCT_LKV
- CZ_SUBTYPERESOURCE_LKV
- CZ_SUBTYPETOTAL_LKV
- CZ_UCTMESSAGETYPE_LKV
- CZ_UCT_PARNTCONTTY_LKV
- CZ_UI_HGRID_ACTIONS_LKV

- CZ_UI_MSTTMP_BOMCON_UILAY_LKV
- CZ_UI_MSTTMP_CNTRLLAYOUT_LKV
- CZ_UI_MSTTMP_NBOMCON_UILAY_LKV
- CZ_UI_MSTTMP_PAGINATION_LKV
- CZ_UI_MSTTMP_PAG_CMP_LKV
- CZ_UI_MSTTMP_PAG_DDNCTRL_LKV
- CZ_UI_MSTTMP_PAG_NOC_LKV
- CZ_UI_MSTTMP_PAG_REF_LKV
- CZ_UI_MSTTMP_PRINAV_LKV
- CZ_UI_MSTTMP_SUPDIS_LKV
- CZ_UI_MSTTMP_TMPUSG_LKV
- CZ_UI_MSTTMP_TMPUSG_MSGUTL_LKV
- CZ_USAGES_PICKER_V
- CZ_VALIDRESULTFORCOMPON_LKV
- CZ_VALIDRESULTFOROPTFEA_LKV
- CZ_VERTICALALIGNMENT_LKV
- CZ_VIEWBYSELECTION_LKV

RULE Rule Tables

These tables hold Rule information and information on the participants in a rule.

- CZ_COMBO_FEATURES
- CZ_COMPATCELL_NODE_V
- CZ_DES_CHART_CELLS
- CZ_DES_CHART_COLUMNS
- CZ_DES_CHART_FEATURES

- CZ_EXPRESSION_NODES
- CZ_FILTER_SETS
- CZ_GRID_CELLS
- CZ_GRID_COLS
- CZ_GRID_DEFS
- CZ_IMP_RULES
- CZ_MODELRULEFOLDER_IMAGES_V
- CZ_MODEL_ALL_RULEFOLDERS_V
- CZ_NODETYPE_SYSPROPS_V
- CZ_NODE_USAGE_IN_RULES_V
- CZ_PSN_TYPED_RULE_REFS_V
- CZ_RULES
- CZ_RULES_WITH_ARGS_V
- CZ_RULETEMPLS_BYLABEL_V
- CZ_RULE_EXPRDETLS_V
- CZ_RULE_EXPRESSION_V
- CZ_RULE_FOLDERS
- CZ_RULE_PARTICIPANTS_V
- CZ_RUL_TYPEDPSN_V
- CZ_TYPED_RULES_V

TXT - Text Tables

These tables store the text that is displayed during runtime Configurator as well as MLS information.

- CZ_IMP_LOCALIZED_TEXTS
- CZ_LOCALIZED_TEXTS

TYP - Data Typing

These tables store the various types of Model nodes, the structure of rule templates, and the elements contained in generated User Interfaces.

- CZ_DATA_SUBTYPES_V
- CZ_NODETYPE_PROPERTIES_V
- CZ_NODE_DISPCOND_PROPERTIES_V
- CZ_PARENT_CHILD_RELS_V
- CZ_TYPE_RELATIONSHIPS
- CZ_VALID_RESULT_TYPES_V

UI User Interface Tables

These tables store information that is used in the User Interfaces, such as image information, UI actions, messages, User Interface references, and so on.

- CZ_JRAD_CHUNKS
- CZ_PS_UI_CTRL_MAPS
- CZ_PSNODETYPE_IMAGES_V
- CZ_RULETYPE_IMAGES_V
- CZ_UIDEF_SIGNATURE_TEMPLS_V
- CZ_UIELEMENT_IMAGES_V
- CZ_UITEMPLS_FOR_PSNODES_V
- CZ_UITEMPL_CONTROLS_V
- CZ_UITEMPL_MESSAGES_V
- CZ_UITEMPL_UTILITY_V
- CZ_UI_ACTIONS
- CZ_UI_COLLECT_TMPLS_V
- CZ_UI_CONT_TYPE_TEMPLS

- CZ_UI_CONT_TYPE_TEMPLS_VV
- CZ_UI_DEFS
- CZ_UI_ELEMENT_ATTRIBUTES_V
- CZ_UI_IMAGES
- CZ_UI_NODES
- CZ_UI_NODE_PROPS
- CZ_UI_PAGES
- CZ_UI_PAGE_ELEMENTS
- CZ_UI_PAGE_REFS
- CZ_UI_PAGE_SETS
- CZ_UI_PATHED_IMAGES_V
- CZ_UI_PROPERTIES
- CZ_UI_REFS
- CZ_UI_REF_TEMPLATES
- CZ_UI_TEMPLATES_VV
- CZ_UI_TEMPLATES
- CZ_UI_TYPEDPSN_V
- CZ_UI_XMLS

XFR Transfer Specifications and Control Tables

These tables contain information that is used during import.

- CZ_XFR_FIELDS
- CZ_XFR_PROJECT_BILLS
- CZ_XFR_RUN_INFOS
- CZ_XFR_RUN_RESULTS

- CZ_XFR_STATUS_CODES
- CZ_XFR_TABLES

E

Code Examples

This appendix covers the following topics:

- Overview
- Pricing and ATP Callback Procedures
- Implementing a Return URL Servlet

Overview

This chapter contains code examples that support other chapters of this document. These examples are more complete and longer than the examples provided in the rest of this document, which are often fragments. See the cited background sections for details.

The following table lists the code examples provided in this chapter.

Code Examples Provided

Purpose of Example	Example
Pricing and ATP Callback Procedures, page E-2	Example of Multiple-item Callback Pricing Procedure, page E-2 Example of Callback ATP Procedure, page E-3
Implementing a Return URL Servlet, page E-3	Example Return URL Servlet (Checkout.java), page E-5

You should consult these other documents for details on the tasks described in this section:

- For information on how to write and compile Configurator Extensions, and on how to incorporate them into your configuration model, see the *Oracle Configurator*

Extensions and Interface Object Developer's Guide.

- For information on how to install Configurator Extensions, see the *Oracle Configurator Installation Guide*.
- For an explanation of building a configuration model and updating configurations, see the *Oracle Configurator Developer User's Guide*.

Pricing and ATP Callback Procedures

This appendix contains minimal examples of PL/SQL procedures you might write to use the OC callback interface for pricing and **ATP** procedures.

See the following sections for background:

- Pricing and ATP in Oracle Configurator , page 13-1
- Pricing Callback Interface , page 13-4
- ATP Callback Interface, page 13-8
- Pricing Parameters, page 9-15
- ATP Parameters, page 9-15

Example of Multiple-item Callback Pricing Procedure

Example

```
PROCEDURE price_multiple_items (p_configurator_session_key IN VARCHAR2,
                                p_price_type IN VARCHAR2,
                                p_total_price OUT NUMBER) AS
BEGIN
  update cz_pricing_structures set list_price = 3.0*seq_nbr,
    selling_price = 2.0*seq_nbr,
    where configurator_session_key =
    p_configurator_session_key;
-- calculation using pricing table for storage
    select sum(selling_price) into p_total_price from
cz_pricing_structures
    where configurator_session_key = p_configurator_session_key;
  -- hard-coded price amount
  -- p_total_price := 343.00;
END price_multiple_items;
```

Example of Callback ATP Procedure
Example

```
PROCEDURE call_atp (p_config_session_key IN VARCHAR2,
                    p_warehouse_id IN NUMBER,
                    p_ship_to_org_id IN NUMBER,
                    p_customer_id IN NUMBER,
                    p_customer_site_id IN NUMBER,
                    p_requested_date IN DATE,
                    p_ship_to_group_date OUT DATE) IS
  BEGIN     update cz_atp_requests set ship_to_date = sysdate-10
      where configurator_session_key
      = p_config_session_key;
    p_ship_to_group_date := sysdate;
  END call_atp;
```

Implementing a Return URL Servlet

The first step in implementing a return URL is to register an alias name for the return URL servlet. For details, see the *Oracle Configurator Installation Guide.*

The section Example Return URL Servlet (Checkout.java), page E-5 shows the complete source code for `Checkout.java`, which you can use as a template for constructing your own return URL servlet.

The Java servlet shown here obtains the value of the `valid_configuration` element from the configuration outputs element of the termination message and displays it in the place of the Oracle Configurator window after the end user has closed the window and saved the configuration session.

See the following sections for background:

- The Return URL, page 10-13
- Session Termination, page 10-1
- Submission, page 10-4
- Configuration Status, page 10-5
- Configuration Outputs, page 10-8

The parts of the code that you should customize to work with a configuration output element other than `valid_configuration` are typographically emphasized.

Note the use of `top.location` in the example which causes the servlet output to replace the contents of the runtime Oracle Configurator window.

Note that this example places `Checkout.java` in a package `myorg.myservlets`, which requires the addition of path information to the following line:

```
out.println("top.location = \"/myorg/myservlets/Checkout?ValidConfig=" +
validConfig + "\"");
```

For more information, see the *Oracle Configurator Installation Guide* section on registering

a return URL servlet.

Example Return URL Servlet (Checkout.java)

Example

```
package myorg.myservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.apps.cz.common.XmlUtil;
import oracle.xml.parser.v2.XMLDocument;
import org.xml.sax.SAXException;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
public class Checkout extends HttpServlet {
  // Responds to the UiServlet request containing the <terminate> XML
message
  public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String terminateString = request.getParameter("XMLmsg");
    XMLDocument terminateDoc;
    try {
      terminateDoc = XmlUtil.parseXmlString(terminateString);
    } catch (SAXException se) {
      throw new ServletException(se.getMessage());
    }
    String validConfig = getValidConfig(terminateDoc);
    System.err.println("configuration valid?: " + validConfig);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<script language=\"javascript\">");
    out.println("top.location = \"/myorg/myservlets/Checkout?ValidConfig
=" + validConfig + "\"");
    out.println("</script>");
    out.println("</html>");
  }
  // Responds to the secondary request for the page to replace the
content frame
  // containing the ValidConfig
  public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String validConfig = request.getParameter("ValidConfig");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Checked Out with Valid Configuration
</title></head>");
    out.println("<body>");
    out.println("Configuration Valid?: " + validConfig);
    out.println("</body>");
    out.println("</html>");
  }
  String getValidConfig(XMLDocument doc) {
    return getTagValue(doc, "valid_configuration", null); // get element
from termination msg
  }
  String getTagValue(XMLDocument doc, String tagName, String
defaultValue) {    Node n = doc.getDocumentElement();
    if (n != null) {
      NodeList nl = n.getChildNodes();
      if (nl != null) {
        for (int i = 0; i < nl.getLength(); i++) {
```

```
Node cn = nl.item(i);
          if (cn.getNodeName().equals(tagName)) {
            NodeList cnl = cn.getChildNodes();
            if (cnl != null) {
              return cnl.item(0).getNodeValue();
            }
          }
        }
      }
    }
    return defaultValue;
  }
}
```

Glossary

This glossary contains definitions relevant to working with Oracle Configurator.

A

Archive Path

The ordered sequence of Configurator Extension Archives for a Model that determines which Java classes are loaded for Configurator Extensions and in what order.

B

base node

The node in a Model that is associated with a Configurator Extension Rule. Used to determine the event scope for a Configurator Extension.

batch validation

A background process for validating selections in a configuration.

binding

Part of a Configurator Extension Rule that associates a specified event with a chosen method of a Java class. *See also* event.

BOM item

The node imported into Oracle Configurator Developer that corresponds to an Oracle Bills of Material item. Can be a BOM Model, BOM Option Class node, or BOM Standard Item node.

BOM Model

A model that you import from Oracle Bills of Material into Oracle Configurator Developer. When you import a BOM Model, effective dates, ATO (Assemble To Order) rules, and other data are also imported into Configurator Developer. In Configurator Developer, you can extend the structure of the BOM Model, but you cannot modify the BOM Model itself or any of its attributes.

BOM Model node

The imported node in Oracle Configurator Developer that corresponds to a BOM Model created in Oracle Bills of Material.

BOM Option Class node

The imported node in Oracle Configurator Developer that corresponds to a BOM Option Class created in Oracle Bills of Material.

BOM Standard Item node

The imported node in Oracle Configurator Developer that corresponds to a BOM Standard Item created in Oracle Bills of Material.

Boolean Feature

An element of a component in the Model that has two options: true or false.

C

CDL (Constraint Definition Language)

A language for entering configuration rules as text rather than assembling them interactively in Oracle Configurator Developer. CDL can express more complex constraining relationships than interactively defined configuration rules can.

The CIO is the API that supports creating and navigating the Model, querying and modifying selection states, and saving and restoring configurations.

CIO (Oracle Configuration Interface Object)

A server in the runtime application that creates and manages the interface between the client (usually a user interface) and the underlying representation of model structure and rules in the generated logic.

command event

An event that is defined by a character string and detected by a command listener.

Comparison Rule

An Oracle Configurator Developer rule type that establishes a relationship to determine the selection state of a logical Item (Option, Boolean Feature, or List-of-Options Feature) based on a comparison of two numeric values (numeric Features, Totals, Resources, Option counts, or numeric constants). The numeric values being compared can be computed or they can be discrete intervals in a continuous numeric input.

Compatibility Rule

An Oracle Configurator Developer rule type that establishes a relationship among Features in the Model to control the allowable combinations of Options. *See also*,

Property-based Compatibility Rule.

Compatibility Table

A kind of Explicit Compatibility Rule. For example, a type of compatibility relationship where the allowable combination of Options are explicitly enumerated.

component

A piece of something or a configurable element in a model such as a BOM Model, Model, or Component.

Component

An element of the model structure, typically containing Features, that is configurable and instantiable. An Oracle Configurator Developer node type that represents a configurable element of a Model.

Component Set

An element of the Model that contains a number of instantiated Components of the same type, where each Component of the set is independently configured.

configuration

A specific set of specifications for a product, resulting from selections made in a runtime configurator.

configuration attribute

A characteristic of an item that is defined in the host application (outside of its inventory of items), in the Model, or captured during a configuration session. Configuration attributes are inputs from or outputs to the host application at initialization and termination of the configuration session, respectively.

configuration model

Represents all possible configurations of the available options, and consists of model structure and rules. It also commonly includes User Interface definitions and Configurator Extensions. A configuration model is usually accessed in a runtime Oracle Configurator window. *See also* model.

configuration rule

A Logic Rule, Compatibility Rule, Comparison Rule, Numeric Rule, Design Chart, Statement Rule, or Configurator Extension rule available in Oracle Configurator Developer for defining configurations. *See also* rules.

configuration session

The time from launching or invoking to exiting Oracle Configurator, during which end users make selections to configure an orderable product. A configuration session is

limited to one configuration model that is loaded when the session is initialized.

configurator

The part of an application that provides custom configuration capabilities. Commonly, a window that can be launched from a host application so end users can make selections resulting in valid configurations. *Compare* Oracle Configurator.

Configurator Developer

See OCD.

Configurator Extension

An extension to the configuration model beyond what can be implemented in Configurator Developer.

A type of configuration rule that associates a node, Java class, and event binding so that the rule operates when an event occurs during a configuration session.

A Java class that provides methods that can be used to perform configuration actions.

Configurator Extension Archive

An object in the Repository that stores one or more compiled Java classes that implement Configurator Extensions.

connectivity

The connection across components of a model that allows modeling such products as networks and material processing systems.

Connector

The node in the model structure that enables an end user at runtime to connect the Connector node's parent to a referenced Model.

Constraint Definition Language

See CDL

Container Model

A type of BOM Model that you import from Oracle Bills of Material into Oracle Configurator Developer to create configuration models that support connectivity and contain trackable components. Configurations created from Container Models can be tracked and updated in Oracle Install Base

Contributes to

A relation used to create a specific type of Numeric Rule that accumulates a total value. *See also* Total.

Consumes from

A relation used to create a specific type of Numeric Rule that decrements a total value, such as specifying the quantity of a Resource used.

count

The number or quantity of something, such as selected options. *Compare* instance.

CZ

The product shortname for Oracle Configurator in Oracle Applications.

CZ schema

The implementation version of the standard runtime Oracle Configurator data-warehousing schema that manages data for the configuration model. The implementation schema includes all the data required for the runtime system, as well as specific tables used during the construction of the configurator.

D

default

In a configuration, the automatic selection of an option based on the preselection rules or the selection of another option.

Defaults relation

An Oracle Configurator Developer Logic Rule relation that determines the logic state of Features or Options in a default relation to other Features and Options. For example, if A Defaults B, and you select A, B becomes Logic True (selected) if it is available (not Logic False).

Design Chart

An Oracle Configurator Developer rule type for defining advanced Explicit Compatibilities interactively in a table view.

E

element

Any entity within a model, such as Options, Totals, Resources, UI controls, and components.

end user

The ultimate user of the runtime Oracle Configurator. The types of end users vary by project but may include salespeople or distributors, administrative office staff, marketing personnel, order entry personnel, product engineers, or customers directly

accessing the application via a Web browser or kiosk. *Compare* user.

event

An action or condition that occurs in a configuration session and can be detected by a listener. Example events are a change in the value of a node, the creation of a component instance, or the saving of a configuration. The part of model structure inside which a listener listens for an event is called the event binding scope. The part of model structure that is the source of an event is called the event execution scope. *See also* command event.

Excludes relation

An Oracle Configurator Developer Logic Rule type that determines the logic state of Features or Options in an excluding relation to other Features and Options. For example, if A Excludes B, and if you select A, B becomes Logic False, since it is not allowed when A is true (either User or Logic True). If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or Unknown. *See* Negates relation.

F

feature

A characteristic of something, or a configurable element of a component at runtime.

Feature

An element of the model structure. Features can either have a value (numeric or Boolean) or enumerated Options.

G

generated logic

The compiled structure and rules of a configuration model that is loaded into memory on the Web server at configuration session initialization and used by the Oracle Configurator engine to validate runtime selections. The logic must be generated either in Oracle Configurator Developer or programmatically in order to access the configuration model at runtime.

guided buying or selling

Needs assessment questions in the runtime UI to guide and facilitate the configuration process. Also, the model structure that defines these questions. Typically, guided selling questions trigger configuration rules that automatically select some product options and exclude others based on the end user's responses.

H

host application

An application within which Oracle Configurator is embedded as integrated functionality, such as Order Management or *i*Store.

I

implementer

The person who uses Oracle Configurator Developer to build the model structure, rules, and UI customizations that make up a runtime Oracle Configurator. Commonly also responsible for enabling the integration of Oracle Configurator in a host application.

Implies relation

An Oracle Configurator Developer Logic Rule type that determines the logic state of Features or Options in an implied relation to other Features and Options. For example, if A Implies B, and you select A, B becomes Logic True. If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or Unknown. *See* Requires relation.

import server

A database instance that serves as a source of data for Oracle Configurator's Populate, Refresh, Migrate, and Synchronization concurrent processes. The import server is sometimes referred to as the remote server.

initialization message

The XML (Extensible Markup Language) message sent from a host application to the Oracle Configurator Servlet, containing data needed to initialize the runtime Oracle Configurator. *See also* termination message.

instance

A runtime occurrence of a component in a configuration that is determined by the component node's Instance attribute specifying a minimum and maximum value. *See also* instantiate. *Compare* count.

Also, the memory and processes of a database.

instantiate

To create an instance of something. Commonly, to create an instance of a component in the runtime user interface of a configuration model.

item

A product or part of a product that is in inventory and can be delivered to customers.

Item

A Model or part of a Model that is defined in the Item Master. Also data defined in Oracle Inventory.

Item Master

Data stored to structure the Model. Data in the CZ schema Item Master is either entered manually in Oracle Configurator Developer or imported from Oracle Applications or a legacy system.

Item Type

Data used to classify the Items in the Item Master. Item Catalogs imported from Oracle Inventory are Item Types in Oracle Configurator Developer.

L

listener

A class in the CIO that detects the occurrence of specified events in a configuration session.

Logic Rule

An Oracle Configurator Developer rule type that expresses constraint among model elements in terms of logic relationships. Logic Rules directly or indirectly set the logical state (User or Logic True, User or Logic False, or Unknown) of Features and Options in the Model.

There are four primary Logic Rule relations: Implies, Requires, Excludes, and Negates. Each of these rules takes a list of Features or Options as operands. *See also* Implies relation, Requires relation, Excludes relation, and Negates relation.

M

model

A generic term for data representing products. A model contains elements that correspond to items. Elements may be components of other objects used to define products. A configuration model is a specific kind of model whose elements can be configured by accessing an Oracle Configurator window.

Model

The entire hierarchical "tree" view of all the data required for configurations, including model structure, variables such as Resources and Totals, and elements in support of intermediary rules. Includes both imported BOM Models and Models created in Configurator Developer. May consist of BOM Option Classes and BOM Standard Items.

model structure

Hierarchical "tree" view of data composed of elements (Models, Components, Features, Options, BOM Models, BOM Option Class nodes, BOM Standard Item nodes, Resources, and Totals). May include reusable components (References).

N

Negates relation

A type of Oracle Configurator Developer Logic Rule type that determines the logic state of Features or Options in a negating relation to other Features and Options. For example, if one option in the relationship is selected, the other option must be Logic False (not selected). Similarly, if you deselect one option in the relationship, the other option must be Logic True (selected). *Compare* Excludes relation.

node

The icon or location in a Model tree in Oracle Configurator Developer that represents a Component, Feature, Option or variable (Total or Resource), Connector, Reference, BOM Model, BOM Option Class node, or BOM Standard Item.

Numeric Rule

An Oracle Configurator Developer rule type that expresses constraint among model elements in terms of numeric relationships. *See also*, Contributes to and Consumes from.

O

object

Entities in Oracle Configurator Developer, such as Models, Usages, Properties, Effectivity Sets, UI Templates, and so on. *See also* element.

OCD

See Oracle Configurator Developer.

option

A logical selection made in the Model Debugger or a runtime Oracle Configurator by the end user or a rule when configuring a component.

Option

An element of the Model. A choice for the value of an enumerated Feature.

Oracle Configurator

The product consisting of development tools and runtime applications such as the CZ schema, Oracle Configurator Developer, and runtime Oracle Configurator. Also the

runtime Oracle Configurator variously packaged for use in networked or Web deployments.

Oracle Configurator Developer

The tool in the Oracle Configurator product used for constructing and maintaining configuration models.

Oracle Configurator engine

The part of the Oracle Configurator product that uses configuration rules to validate runtime selections. Compare generated logic. *See also* generated logic.

Oracle Configurator schema

See CZ schema.

Oracle Configurator Servlet

A Java servlet that participates in rendering legacy user interfaces for Oracle Configurator.

Oracle Configurator window

The user interface that is launched by accessing a configuration model and used by end users to make the selections of a configuration.

P

Populator

An entity in Oracle Configurator Developer that creates Component, Feature, and Option nodes from information in the Item Master.

Property

A named value associated with a node in the Model or the Item Master. A set of Properties may be associated with an Item Type. After importing a BOM Model, Oracle Inventory Catalog Descriptive Elements are Properties in Oracle Configurator Developer.

Property-based Compatibility Rule

An Oracle Configurator Developer Compatibility Rule type that expresses a kind of compatibility relationship where the allowable combinations of Options are specified implicitly by relationships among Property values of the Options.

publication

A unique deployment of a configuration model (and optionally a user interface) that enables a developer to control its availability from host applications such as Oracle Order Management or *i*Store. Multiple publications can exist for the same configuration

model, but each publication corresponds to only one Model and User Interface.

publishing

The process of creating a publication record in Oracle Configurator Developer, which includes specifying applicability parameters to control runtime availability and running an Oracle Applications concurrent process to copy data to a specific database.

R

reference

The ability to reuse an existing Model or Component within the structure of another Model (for example, as a subassembly).

Reference

An Oracle Configurator Developer node type that denotes a reference to another Model.

Repository

Set of pages in Oracle Configurator Developer that contains areas for organizing and maintaining Models and shared objects in a single location.

Requires relation

An Oracle Configurator Developer Logic Rule relationship that determines the logic state of Features or Options in a requirement relation to other Features and Options. For example, if A Requires B, and if you select A, B is set to Logic True (selected). Similarly, if you deselect A, B is set to Logic False (deselected). *See* Implies relation.

Resource

A variable in the Model used to keep track of a quantity or supply, such as the amount of memory in a computer. The value of a Resource can be positive or zero, and can have an Initial Value setting. An error message appears at runtime when the value of a Resource becomes negative, which indicates it has been over-consumed. Use Numeric Rules to contribute to and consume from a Resource.

Also a specific node type in Oracle Configurator Developer. *See also* node.

rules

Also called business rules or configuration rules. In the context of Oracle Configurator and CDL, a rule is not a business rule. Constraints applied among elements of the product to ensure that defined relationships are preserved during configuration. Elements of the product are Components, Features, and Options. Rules express logic, numeric parameters, implicit compatibility, or explicit compatibility. Rules provide preselection and validation capability in Oracle Configurator.

See also Comparison Rule, Compatibility Rule, Design Chart, Logic Rule and Numeric Rule.

runtime

The environment in which an implementer (tester), end user, or customer configures a product whose model was developed in Oracle Configurator Developer. *See also* configuration session.

S

Statement Rule

An Oracle Configurator Developer rule type defined by using the Oracle Configurator Constraint Definition Language (text) rather than interactively assembling the rule's elements.

T

termination message

The XML (Extensible Markup Language) message sent from the Oracle Configurator Servlet to a host application after a configuration session, containing configuration outputs. *See also* initialization message.

Total

A variable in the Model used to accumulate a numeric total, such as total price or total weight.

Also a specific node type in Oracle Configurator Developer. *See also* node.

U

UI

See User Interface.

UI Templates

Templates available in Oracle Configurator Developer for specifying UI definitions.

Unknown

The logic state that is neither true nor false, but unknown at the time a configuration session begins or when a Logic Rule is executed. This logic state is also referred to as Available, especially when considered from the point of view of the runtime Oracle Configurator end user.

user

The person using a product or system. Used to describe the person using Oracle Configurator Developer tools and methods to build a runtime Oracle Configurator. *Compare* end user.

user interface

The visible part of the application, including menus, dialog boxes, and other on-screen elements. The part of a system where the user interacts with the software. Not necessarily generated in Oracle Configurator Developer. *See also* User Interface.

User Interface

The part of an Oracle Configurator implementation that provides the graphical views necessary to create configurations interactively. A user interface is generated from the model structure. It interacts with the model definition and the generated logic to give end users access to customer requirements gathering, product selection, and any extensions that may have been implemented. *See also* UI Templates.

V

validation

Tests that ensure that configured components will meet specific criteria set by an enterprise, such as that the components can be ordered or manufactured.

W

Workbench

Set of pages in Oracle Configurator Developer for creating, editing, and working with Repository objects such as Models and UI Templates.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X