Oracle® Configurator

Implementation Guide

Release 11*i*

Part No. B10615-01

Februrary 2003

This book provides explanations, description, and instructions for the administration tasks required to set up and support development and deployment of a runtime Oracle Configurator.

ORACLE®

Oracle Configurator Implementation Guide, Release 11*i*

Part No. B10615-01

Primary Author: Tina Brand, Stephen Damiani, Mark Sawtelle, Harriet Shanzer

Contents

4 The CZ Schema

5 Populating the CZ Schema

10 Session Termination

11 Batch Validation

12 Custom Integration

13 Pricing and ATP in Oracle Configurator

14 Multiple Language Support

Part IV Configuration Model

15 Controlling the Development Environment

16 Customizing the HTML Template

17 Publishing Configuration Models

18 Programmatic Tools for Development

19 Programmatic Tools for Maintenance

A Common Tasks

B Concurrent Programs

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Configurator Implementation Guide, Release 11*i*

Part No. B10615-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: czdoc_us@oracle.com
- FAX: 781-238-9898. Attn: Oracle Configurator Documentation
- Postal service:
  Oracle Corporation
  Oracle Configurator Documentation
  10 Van de Graaff Drive
  Burlington, MA 01803- 5146
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual presents tasks and information useful in implementing Oracle Configurator, including information formerly covered in the *Oracle Configurator Custom Web Deployment Guide*.

See the *Oracle Configurator Installation Guide* for installation information and the *Oracle Configurator Developer User's Guide* for information about developing configuration models in Oracle Configurator Developer.

Intended Audience

Anyone responsible for supporting use of Oracle Configurator should read this book. That 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.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

This manual contains a table of contents, lists of examples, tables and figures, a reader comment form, several chapters, appendix, glossary, and index. The chapters are organized in five parts. Within the chapters, information is organized in numbered sections of several levels. Note that level does not imply importance or degree of detail. For instance, third-level sections in one chapter (x.x.x) may not contain information of equivalent detail to the third-level sections in another chapter.

- Part I, "Introduction"
  - Chapter 1, "Implementation Tasks" presents an overview of all known tasks in an Oracle Configurator implementation, including custom tasks.
  - Chapter 2, "Configurator Architecture" describes the elements of the Oracle Configurator product and how they fit together.
- Part II, "Data"
  - Chapter 3, "Database Instances" describes the uses to which databases are put when implementing Oracle Configurator, and specifics about using multiple database instances.
  - Chapter 4, "The CZ Schema" describes the basic characteristics of the Oracle Configurator schema, the schema settings and how they are used, and provides some schema maintenance tips.
  - Chapter 5, "Populating the CZ Schema" 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 Oracle Configurator schema, data

import verification, the process for refreshing or updating imported data, and customizing data import.

  - Chapter 6, "Migrating Data" describes how to transfer data from one database instance to another. This includes transferring data from a standalone Oracle SellingPoint schema to an Oracle 11*i* schema as well as transferring data from one Oracle Configurator 11*i* schema to another.
  - Chapter 7, "Synchronizing Data" describes when and how data should be synchronized. This includes: synchronizing BOM data after the import server has changed, and synchronizing publication data after a database has been cloned.
  - Chapter 8, "CZ Schema Maintenance" explains how to maintain data when it exists in more than one place and is potentially unsynchronized.

- Part III, "Integration"
  - Chapter 9, "Session Initialization" describes the format and parameters of the initialization message for the Oracle Configurator Servlet.
  - Chapter 10, "Session Termination" describes the format and parameters of the termination message for the Oracle Configurator Servlet.
  - Chapter 11, "Batch Validation" describes using Oracle Configurator in a programmatic mode.
  - Chapter 12, "Custom Integration" explains how to modify certain Oracle Configurator files as well as the purpose of the files and where they can be found.
  - Chapter 13, "Pricing and ATP in Oracle Configurator" provides an overview of how pricing works in a runtime Oracle Configurator.
  - Chapter 14, "Multiple Language Support" explains how you can modify Item descriptions in Oracle Applications and have them appear when you develop configuration models and deploy User Interfaces.
- Part IV, "Configuration Model"
  - Chapter 15, "Controlling the Development Environment" describes how you can control the Oracle Configurator Developer environment with profile options and the `spxi.ini` file.
  - Chapter 16, "Customizing the HTML Template" describes some ways of customizing the HTML Template files that comprise part of the user interface for the runtime Oracle Configurator. This chapter does not explain

how to customize a user interface that is not generated through Configurator Developer, or that uses a non-Oracle host application.

  - Chapter 17, "Publishing Configuration Models" explains the database processes for publishing configuration models to make them available to host applications.
  - Chapter 18, "Programmatic Tools for Development" 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.
  - Chapter 19, "Programmatic Tools for Maintenance" describes a set of programmatic tools (PL/SQL procedures) that you can use primarily to maintain a deployed runtime Oracle Configurator.

- Part V, "Runtime Configurator"
  - Chapter 20, "User Interface Deployment" 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.
  - Chapter 21, "Deployment Considerations" describes the strategies you should consider when you are ready to complete the deployment of a runtime Oracle Configurator.
  - Chapter 22, "Managing Configurations" describes the data structures produced by Oracle Configurator during a configuration session.
- Part VI, "Appendices"
  - Appendix A, "Common Tasks" describes certain tasks that may be required while implementing an Oracle Configurator. These tasks include: running concurrent programs, server administration, connecting to a database instance, verifying the Configurator schema version, viewing status of Configurator concurrent programs, querying registered Configurator concurrent programs, checking BOM and Model Similarity.
  - Appendix B, "Concurrent Programs" describes the concurrent programs available to either the Configurator Administrator or Configurator Developer responsibility.
  - Appendix C, "CZ Subschemas" lists the CZ tables that make up each of the subschemas in the CZ schema. For table details, see Configurator eTRM on Metalink, Oracle's technical support Web site.

  - Appendix D, "Code Examples" contains code examples that support other chapters of this document. These examples are fuller and longer than the examples provided in the rest of this document, which are often fragments.

- "Glossary of Terms and Acronyms" contains definitions that you may need while working with Oracle Configurator documentation.

  The Index provides an alternative method of searching for key concepts and product details.

Related Documents

The following documents are also included in the Oracle Configurator documentation set on the Oracle Configurator Developer compact disc:

- *Oracle Configurator Release Notes*
- *Oracle Configurator Installation Guide*
- *Oracle Configurator Developer User's Guide*
- *Oracle Configuration Interface Object (CIO) Developer's Guide*
- *Oracle Configurator Performance Guide*
- *Oracle Configurator Methodologies*

For more information, see the documentation for Oracle Applications (Release 11*i*) Oracle RDBMS (Release 8*i* or 9*i*), the *Oracle Applications Library*, the product-specific Release Notes for releases supported to work with Oracle Configurator, and the Configurator eTRM on Metalink, Oracle's technical support Web site.

Additionally, as useful background in implementing applications, consult:

- *Oracle9i Database Performance Methods*

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 following conventions are also used in this manual:

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

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.

To report issues that are not mission-critical, submit a Technical Assistance Request (TAR) using Metalink, Oracle's technical support Web site at:

`http://www.oracle.com/support/metalink/`

Log into your Metalink account and navigate to the Configurator TAR template:

1. Choose the **TARs** link in the left menu.
2. Click on **Create a TAR**.

3. Fill in or choose a profile.
4. In the same form:
   a. Choose **Product**: Oracle Configurator or Oracle Configurator Developer
   b. Choose **Type of Problem**: Oracle Configurator Generic Issue template
5. Provide the information requested in the iTAR template.

You can also find product-specific documentation and other useful information using Metalink.

For a complete listing of available Oracle Support Services and phone numbers, see:

`www.oracle.com/support/`

Part I

Introduction

Part I consists of chapters that present a baseline for understanding Oracle Configurator. The chapters are:

- Chapter 1, "Implementation Tasks"
- Chapter 2, "Configurator Architecture"

1

Implementation Tasks

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

- General Implementation Tasks
- Database Tasks
- Integration Tasks
- Model Development Tasks
- Deployment Tasks

1.1 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 and Oracle Configurator schema. See the *Oracle Configurator Installation Guide* for additional information.
- Configure the workstation, Oracle Configurator Developer, JInitiator, and your browser to display appropriate fonts for Multiple Language Support (MLS). See the *Oracle Configurator Installation Guide* for details.
- Install or upgrade Oracle Configurator Developer to the desired release or patch level. See the *Oracle Configurator Installation Guide* for details.
- Check the *Oracle Configurator Release Notes* and Metalink for any effects an Oracle Configurator Developer upgrade may have on your development and test environments; new functionality in Configurator Developer may depend on other applications.

- Assign a Configurator responsibility (either the Configurator Administrator or Configurator Developer) in order for a user to run the Configurator concurrent programs. For more information about assigning responsibilities, see the *Oracle Applications System Administrator's Guide.*

1.2 Database Tasks

Database tasks are those tasks that setup and support the Oracle Configurator schema (CZ), which is part of the Oracle Applications database.

1.2.1 Required Database Tasks

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

- Decide which database instance is for production and which, if different, are for development and testing. For information see Chapter 3, "Database Instances".
- Verify that Inventory and BOM model data in Oracle Applications is correctly defined. See Section 5.2, "Standard Import".
- 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 Oracle Configurator schema. For information, see Section 5.3, "Custom Import"
- 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 Section 5.2.6, "Exploding BOMs in Oracle Applications".
- Define and enable servers, as needed for data import, synchronization, and publishing. For information, see Section B.2, "Server Administration Concurrent Programs" on page B-4.
- Modify Configurator Parameters. Running this concurrent program as the Configurator Administrator sets installation-wide customizable settings (CZ_DB_SETTINGS) that describe the structure and content of the Oracle Configurator schema, and define application functions. For information, see Section B.1.2, "Modify Configurator Parameters".
- Populate the CZ schema with production BOM and Inventory data for use in defining configuration models. This is called data import in Oracle Configurator documentation. For information, see Section 5.2, "Standard Import".

- Control the scope of the data import by modifying values in the integration tables (CZ_XFR_) provided for that purpose. For information, see Section 4.3, "Control Tables".
- Refresh data in the Oracle Configurator schema as production BOM and Inventory data changes. For information, see Section 5.2.10, "Refreshing Imported Data".
- Verify that populating or refreshing the Oracle Configurator schema BOM data is correct by viewing the CZ Item Master in Configurator Developer. For information, see the *Oracle Configurator Developer User's Guide*.
- Synchronize BOM data in the Oracle Configurator 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 Section 7.2, "Synchronizing BOM Data".
- Purge tables in the Oracle Configurator schema if your database gets too large and fails to perform adequately. The Purge Configurator Tables concurrent program deletes those records that are marked for deletion. For more information see Section 8.3, "Purging Configurator Tables".

1.2.2 Optional Database Tasks

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

- To make use of Multiple Language Support (MLS), use PL/SQL to modify nodes created in Configurator Developer and BOM item descriptions. For more information see Chapter 14, "Multiple Language Support".

1.3 Integration Tasks

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

1.3.1 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.

1.3.2 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.

- Customize certain HTML template files to modify the appearance of the Oracle Configurator window. For details see Chapter 16, "Customizing the HTML Template".
- Test and install the customized HTML template files. See Section 16.3, "Customizing the HTML Template Files".
- Provide pricing and ATP support for the runtime Oracle Configurator by setting switches in `jserv.properties` file. See Chapter 13, "Pricing and ATP in Oracle Configurator"
- Enable Multiple Language Support (MLS). For details see Chapter 14, "Multiple Language Support".
- Set up the Model structure and Functional Companions for configuration attributes. See *Oracle Configurator Methodologies*.

1.3.3 Tasks for Custom Integration

These tasks (in addition to the required tasks listed in Section 1.3.1) 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 Chapter 9, "Session Initialization"

- Create and install a servlet that handles the runtime Oracle Configurator's XML termination message, which contains configuration output data. For details, seeChapter 10, "Session Termination".
- Set up a Return URL for the servlet that handles the termination message, and add it to the initialization message. For details, seeChapter 9, "Session Initialization".

1.4 Model Development Tasks

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

1.4.1 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.

- Install Oracle Configurator Developer. See the *Oracle Configurator Installation Guide*.
- Edit the default settings of the spx.ini file to specify Configurator Developer behavior and connections to the database and OC Servlet during development, unit testing, and maintenance of configuration models. See Section 15.2, "Spx.ini".
- Design configuration models with performance in mind. See the *Oracle Configurator Performance Guide* for guidelines.
- Verify 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 Section 1.2, "Database Tasks"for additional tasks needed to populate the CZ schema.
- Define the structure, rules, and the user interface of the configuration model. See the *Oracle Configurator Developer User's Guide* for more information.
- Run the Generate Active Model command to create the structure and rules of the configuration model. Rerun this command after you have done the following activities:
  - Changed rule definitions
  - Changed the Model structure

- Run the Create New User Interface command to generate a default User Interface.
- Run the Refresh command to update a User Interface with the latest modifications to the User Interface definitions and customizations. Rerun this command after you have done 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*.
- Define the configuration model's applicability parameters in preparation for publishing the configuration model so that it can be accessed by a hosting application. See the *Oracle Configurator Developer User's Guide*
- Create a publication for the configuration model to appropriate hosting applications. See the *Oracle Configurator Developer User's Guide*.
- Publish configuration models for availability to hosting applications by running concurrent programs for that purpose. For information, see Section B.3, "Configuration Model Publication Concurrent Programs".
- Assign each publication record to one Model and the appropriate usages in order to control when and if the usages are invoked by the hosting applications. See the *Oracle Configurator Developer User's Guide* for additional information.
- Republish the configuration model if the model's structure, rules, UI, or applicability parameters change. See the *Oracle Configurator Developer User's Guide*.

1.4.2 Optional Tasks for Model Development

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

- Write Functional Companions to extend the functional capabilities of your configuration model beyond what can implemented in Oracle Configurator Developer. For information on writing Functional Companions see the *Oracle Configuration Interface Object (CIO) Developer's Guide, Oracle Configurator Methodologies*, and the *Oracle Configurator Developer User's Guide*.

1.4.3 Custom Tasks for Model Development

This task can be performed to provide custom Model functionality.

- Create a custom UI that adheres to the Oracle guidelines in Chapter 16, "Customizing the HTML Template".

1.5 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 hosting application.

1.5.1 Required Tasks for All Deployments

The following task is required in order for the runtime Oracle Configurator to use the Java applet user interface:

- JInitiator 1.1.8.7 must be installed on the client machine. See the *Oracle Configurator Installation Guide*.

The following tasks are required in order for the runtime Oracle Configurator to use the DHTML user interface:

- The browser running the DHTML runtime Oracle Configurator must be set up to enable stylesheets and JavaScript. See either Microsoft Internet Explorer, or Netscape Navigator Help for details.
- Use Microsoft Internet Explorer 4.0 or higher, or Netscape Navigator 4.07 or higher, to best view the DHTML runtime Oracle Configurator. The browser must support frames.
- The browser must be set up to accept and send cookies. See either Microsoft Internet Explorer, or Netscape Navigator Help for details.
- Recommended screen resolution is 800 X 600 or higher. 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.
- Copy images that are used in the DHTML UI to the server. For more information, see Section 16.3.1, "Specifying the Location of Media Files"

1.5.2 Optional Tasks for Deployment

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

- Optimize the performance of the production environment by:
  - Adjusting machine size or setting up the database and application tiers on multiple server machines.
  - Tuning components of the Oracle Configurator architecture on the client machine, such as browser settings, swap space, and memory
  - Adjusting Web server configuration settings

  For details see the *Oracle Configurator Performance Guide*:
- Load balance the Apache Web listener (HTTP). For details see the *Oracle Configurator Installation Guide*.
- Set up Secured Sockets Layer (SSL) if you want to create a secure connection between a client and server machine. See Section 21.1.4, "Implementing Oracle Configurator with Secure Sockets Layer" on page 21-5 for details.
- 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.
- 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. See Section 13.4, "Controlling Pricing in the Runtime Oracle Configurator" on page 13-15. Table 20–1, " Information Details Mapped to Oracle Configurator Documentation" on page 20-2 lists Oracle Configurator Documentation for more information.

1.5.3 Tasks for Custom Deployments

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

- Create a custom UI that adheres to the Oracle guidelines in Chapter 16, "Customizing the HTML Template".
- Create online Help for the runtime Oracle Configurator. See *Oracle Configurator Developer User's Guide* for generic runtime information.

2

Configurator Architecture

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

- Three-Tier Architecture
- Client/Server Development Architecture
- Custom Web Application Architecture

For information about machine architecture and requirements, see the *Oracle Configurator Installation Guide*. For performance considerations, see the *Oracle Configurator Performance Guide*.

2.1 Overview

Oracle Configurator consists of the following main elements:

- Runtime Oracle Configurator
- Oracle Configurator Servlet (OC Servlet)
  - UI Server
  - Configuration Interface Object (CIO)
- The Oracle Configurator Schema within an Oracle Applications database
- Oracle Configurator Developer

The runtime Configurator, Configurator Servlet, and Configurator schema are installed with Oracle Applications Release 11*i*. Oracle Configurator Developer must be installed separately.

2.1.1 Runtime Oracle Configurator

The runtime Configurator provides a means of selecting options to create a configuration. Although this can be a programmatic process within an enterprise, it is most commonly an interactive process within a business application. For example, business events could trigger programmatic selections and validations in a batch mode, or end users could view lists of options and make selections that are validated immediately to ensure that the configured item can be ordered and manufactured.

In the case of an interactive process, the runtime Oracle Configurator is presented to end users by the OC Servlet as an Oracle Configurator window. Although Configurator can be deployed as a standalone application, the Oracle Configurator window is usually launched from within a host application. For a list of Oracle Applications that integrate with Configurator, see the *Oracle Configurator Release Notes.*

The runtime Oracle Configurator works in the following scenarios:

- In Oracle Applications Release 11*i*, with the Java applet or DHTML runtime Oracle Configurator displaying the configuration model in a supported host application.
- In a custom web application, with a DHTML runtime Oracle Configurator for configuring items based on a configuration model.
- For unit testing, a browser with a runtime Oracle Configurator launched from Oracle Configurator Developer.

The runtime Configurator is initialized when end users select a configurable item in a host application and then invoke the Configurator, usually by clicking a button or choosing a menu command.

In forms-based hosting Oracle Applications, when an end user selects a configurable ATO or PTO BOM Model for which no configuration model has been defined (in Oracle Configurator Developer), then the OC Servlet passes data directly from the Oracle Bills of Material tables in the Oracle Applications database and uses the forms-based Java applet to present the runtime Configurator.

If the configurable item is a configuration model defined in Configurator Developer, then the OC Servlet passes data from the Oracle Configurator schema (CZ) tables and uses the UI type and definitions in the configuration model to present the runtime Configurator. The configuration model can be based on an ATO or PTO BOM Model, in which case the CZ tables are populated with the read-only structure and implicit rules that represent the ATO or PTO BOM Model.

For more information about how the runtime Oracle Configurator is rendered on the user interface tier, see Section 2.2.1, "User Interface Tier" on page 2-8.

2.1.2 Oracle Configurator Servlet

The Oracle Configurator Servlet is the machinery responsible for rendering the User Interface and brokering the communication between the configuration model, the database, and the client.

The OC Servlet consists of the following elements:

- UI Server
- Configuration Interface Object (CIO)
- Configuration engine (Active Model)

The OC Servlet runs on Oracle Internet Application Server (*i*AS), 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*.

2.1.2.1 UI Server

The UI Server is an element of the OC Servlet that processes user input from the client's user interface and renders the UI for display to the end user based on information received from the configuration engine. For the DHTML UI, the UI Server generates a JavaScript rendering of the configuration model's structure and rules, based on a User Interface generated and modified with Configurator Developer.

The UI Server provides a common level of support for all user interfaces created in Oracle Configurator Developer (both DHTML and Java applet).

2.1.2.2 Configuration Engine (Active Model)

The configuration engine validates user selections and provides results based on the compiled structure and rules of a configuration model (Active Model).

The configuration engine has no public API and cannot be modified.

2.1.2.3 Configuration Interface Object

The CIO is an API layer that handles communication between the Active Model and the UI. Functional Companions and custom UIs communicate with the Active

Model through the CIO. The API methods of the CIO can be used to access the configuration model and Configurator behaviors.

For more information see the *Oracle Configuration Interface Object (CIO) Developer's Guide*

2.1.3 Oracle Configurator Schema

The Oracle Configurator schema consists of Configurator (CZ) tables in the Oracle Applications 11*i* 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

For more information about the Oracle Configurator schema data model, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

2.1.4 Oracle Configurator Developer

Oracle Configurator Developer (OCD) is the development application used to define the model structure, rules, and UI definitions and customizations of configuration models.

Configurator Developer is a thick client, database-intensive application that connects directly to the CZ schema. The ODBC data source selected by a Configurator Developer user at login identifies the database to which OCD connects.

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

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

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

When unit testing and debugging configuration models using the Configurator Developer Test module, you access the runtime Configurator as a test environment directly from Configurator Developer. The Test module accesses the same application architecture as a deployed runtime Configurator. The test Configurator is rendered in a browser using an Oracle Configurator Servlet. Testing from Configurator Developer does not involve running the host application where your configuration models are deployed, such as Order Management.

2.2 Three-Tier Architecture

The Oracle Configurator elements are organized in a three-tier architecture:

1. User interface, presentation, or client tier
2. Application or processing tier
3. Data tier

The configuration model, user interface and configuration rule definitions are *stored* in the data tier. These definitions are *processed* in the applications tier and *rendered* in the user interface tier. Table 2–1 on page 2-5 lists the Oracle Configurator elements across these tiers.

Table 2–1 Oracle Configurator Elements Across Three Tiers

Tier	Tier Elements	Configurator Elements
user interface	thin client	■ Browser where the runtime Oracle Configurator is rendered ■ Browser where the host application that launches Configurator is rendered
	thick client	■ Oracle Configurator Developer
application	application (Web) server application logic	■ Oracle Configurator Servlet on Apache Web server in Oracle Internet Application Server (*i*AS) ■ Oracle Applications Release 11*i*

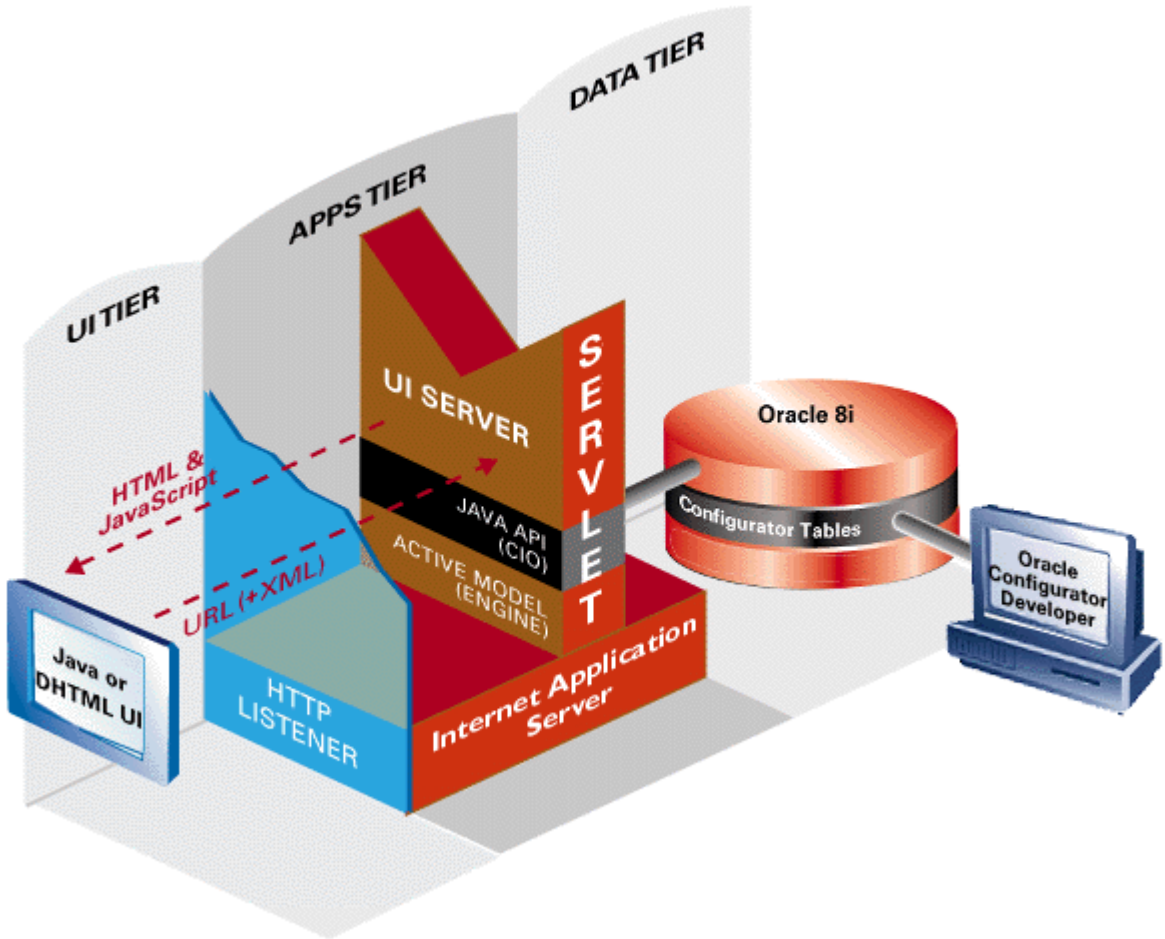
Table 2–1 (Cont.) Oracle Configurator Elements Across Three Tiers

Tier	Tier Elements	Configurator Elements
data	application database	■ Oracle Configurator schema in the Oracle Applications database on Oracle 9*i* or Oracle8*i* Enterprise Edition

A production Oracle Configurator and the test runtime Configurator user interface invoked from Oracle Configurator Developer are thin-client UIs using the same three-tier architecture. The application server is the critical element that enables the thin-client browser to maintain a Web interface to the configuration model accessed by the Oracle Configurator window.

Figure 2–1 on page 2-7 illustrates this architecture and shows Oracle Configurator Developer during model development accessing the database directly.

Figure 2–1 Architectural Overview of Oracle Configurator



Most of Oracle Configurator processing occurs in the Web server portion of the application tier and data retrieved from the database is processed in memory in the Web server.

2.2.1 User Interface Tier

During runtime, the user interface tier contains the thin-client browser in which host applications and the runtime Oracle Configurator are rendered. The runtime Configurator UI can be:

- A default UI generated dynamically from model structure and rules
- A customized UI generated dynamically from model structure and rules
- A custom UI defined to communicate with the Active Model through the CIO

A generated UI dynamically takes its definition from the structure and rules of the configuration model. UI customization is not necessary, though it may be desired if the default UI generated by the Model does not fulfill your business needs.

The generated UI may be one of two types:

- Java applet for basic Assemble to Order (ATO) and Pick to Order (PTO) option selection
- Dynamic HTML (DHTML) for custom look and feel, including guided buying or selling questions

A single configuration model can support multiple user interfaces. Different UI styles may be required by different users or channels, such as:

- Back-office order management
- A front-office Web store
- Business-to-business transactions
- Business-to-consumer transactions

The host application can be set up to determine which UI style to use, based on the information in the Configurator's UI definitions, the publication record of the configuration model, or user permissions. During unit testing from Configurator Developer, users choose the UI from a list after selecting the Test button.

2.2.1.1 Java Applet User Interface

The runtime Java applet UI:

- Has a static look and feel based on PTO or ATO BOM Models, Option Classes and Standard Items
- Cannot be customized, and can be used when guided buying or selling is not required

- Is only available with Oracle Applications forms-based applications and is the default UI for those applications if no DHTML UI is defined
- Is used to configure a PTO or ATO BOM Model for which no configuration model has been defined in OCD.

2.2.1.2 DHTML User Interface

The difference between HTML and DHTML is that while HTML is a fixed markup language for displaying content on the Web, DHTML is a collection of technologies that includes support for modifying Web content (objects on the screen) without repainting the whole page. This is the advantage in presenting a Configurator in DHTML. When the end user makes a selection, only screen objects that have changed are redisplayed.

The runtime DHTML UI:

- Uses configuration-driven JavaScript controls
- Is customizable to reflect corporate branding
- Is used when guided buying or selling is required

Customization includes adding links to external image files (there is no drawing package within Configurator Developer), adding or changing text, fonts, colors, buttons, and modifying screen location of UI controls.

A standard UI template is provided with Oracle Configurator, as a set of JSP and HTML files. Implementers can modify this template by adding more JavaScript controls to enhance functionality. Image files (to support UI customizations) are stored on the server.

2.2.2 Application Tier

The application tier runs Oracle Applications Release 11*i* and Oracle Internet Application Server, which are both installed with Rapid Install. Oracle Applications includes the Oracle Configurator Servlet. For details about the OC Servlet, see Section 2.1.2 on page 2-3.

2.2.3 Data Tier

The data tier runs the Oracle Applications database on Oracle 9*i* or Oracle8*i* Enterprise Edition, either of which can be installed with Rapid Install.

For information about installing Oracle Applications, see the *Installing Oracle Applications* manual. For more information about the CZ schema in the Oracle Applications database, see Section 2.1.3, "Oracle Configurator Schema" on page 2-4.

2.3 Client/Server Development Architecture

During development of structure, rules, and UI definition, the architecture is two-tier client / server with the thick client Oracle Configurator Developer accessing the CZ schema of the Oracle Applications database directly as shown in Figure 2–1 on page 2-7.

As a database intensive application, there are certain requirements and recommendations that critically affect Configurator Developer performance. For instance, Oracle recommends running Configurator Developer on a client Windows machine that is networked by local area network (LAN) to the Release 11*i* Oracle Applications database on a server machine. For more information and other alternatives, see the *Oracle Configurator Installation Guide* and the *Oracle Configurator Release Notes*.

The development architecture used during unit testing is the same as the runtime architecture described in Section 2.2, "Three-Tier Architecture" on page 2-5. The Oracle Configurator Servlet used during unit testing is installed and set up specifically for testing from Configurator Developer and is not shared with deployed Configurators.

For additional information about the development environment, see Section 3.3, "Development Database Instance" on page 3-5 and Chapter 15, "Controlling the Development Environment".

2.4 Custom Web Application Architecture

A custom web application is one that incorporates the runtime Oracle Configurator in an Oracle Configurator window, but which departs from the standard Oracle Configurator architecture in order to satisfy specific requirements not otherwise available.

The major elements of a custom application are listed in Table 2–2. Compare this list to the one in Table 2–1 on page 2-5. Those elements that differ are typographically highlighted, and are explained in Section 2.4.1 on page 2-11.

Table 2–2 Elements of a Custom Web Application Using Oracle Configurator

Tier	Tier Elements	Configurator Elements
user interface	thin client	■ Browser where the runtime Oracle Configurator is rendered ■ Browser where the host application that launches Configurator is rendered ■ **Custom UI code**
	thick client	■ Oracle Configurator Developer ■ **3rd-party development tools for Web UI**
application	application (Web) server application logic	■ Oracle Configurator Servlet on Apache Web server in Oracle Internet Application Server (*i*AS) ■ **Custom host application** ■ **Custom interface code using the CIO**
data	application database	■ Oracle Configurator schema in the Oracle Applications database on Oracle 9*i* or Oracle8*i* Enterprise Edition

A particular custom application may include some or all of the custom elements.

2.4.1 Comparison to Standard Application Architecture

The architecture for a custom application differs from the standard Oracle Configurator architecture in these areas:

- **Custom host application**

  In place of Oracle Applications, you must provide the custom host application itself.

- **Custom UI code**

  In place of the provided JavaScript code that renders the DHTML user interface for the runtime Oracle Configurator, you may need to provide JavaScript or other means for rendering the user interface. You may need to do this if your desired UI requires certain effects that are not currently provided by Oracle Configurator.

- **Custom interface code using the CIO**

In place of the interface between the UI and the Active Model that is built into the runtime Oracle Configurator (the Oracle Configurator UI Server), you must provide custom code that interacts with your UI and with the Active Model; this code must use the Oracle Configuration Interface Object (CIO). You must also provide a custom initialization message from the host application to the runtime Oracle Configurator.

- **3rd-party development tools for Web UI**

  In place of the User Interface module of Oracle Configurator Developer, you must provide a tool for creating and modifying a Web user interface.

Further detail on these custom elements is included in Section 2.4.2, "Required Elements for Custom Applications" on page 2-12.

2.4.2 Required Elements for Custom Applications

Table 2–3 on page 2-12 lists all the elements needed by a Web-based host application that incorporates the Oracle Configurator window, with a brief description of how you use each element in building your host application.

Each required element is marked as being either:

- **Provided by Oracle**, meaning that it is part of the Oracle Configurator installation, and is described in the Oracle Configurator documentation
- **Provided by You**, meaning that you must supply the necessary tools, associated technology, and expertise in their use

Each required element is also marked as being relevant to:

- **Dev**, meaning the development environment for the application
- **Run**, meaning the runtime environment for the application

Table 2–3 Required Elements and Actions for Oracle Configurator Custom Application

Element Required	Provided by...	Dev/Run	Action Required or Provided
Your custom host application (such as a Web store)	You	Run	You implement the application.
Web server (Apache)	You	Dev (Test button) Run	You install and maintain the Apache Web server as part of Oracle Internet Application Server. Otherwise, you must install and maintain a custom Web server.
Oracle Applications database (including Oracle Configurator schema tables)	Oracle	Dev Run	You install the Oracle Applications database You provide connectivity between the database and the OC Servlet.

Table 2–3 (Cont.) Required Elements and Actions for Oracle Configurator Custom Application

Element Required	Provided by...	Dev/Run	Action Required or Provided
Oracle Configurator Servlet	Oracle	Dev (Test button) Run	The OC Servlet encapsulates other OC elements, including: ■ The UI Server ■ The CIO
Oracle Configurator UI Server	Oracle	Dev (Test button) Run	Automatically accessed by OC Servlet to render the user interface.
Oracle Configuration Interface Object (CIO)	Oracle	Dev Run	Automatically accessed by OC Servlet to interact with Active Model. available for custom programming.
Functional Companions	You	Dev Run	You code Java Functional Companions to extend your configuration model.
DHTML Oracle Configurator window	Oracle	Run	Configuration window is produced automatically by OC Servlet.
Configuration window UI layout	Oracle	Run	Layout of UI elements is automatically determined by Model, stored in Oracle Configurator schema.
Default HTML templates for Oracle Configurator window	Oracle	Run	Configuration window automatically displays UI in default format.
Customized HTML templates for Oracle Configurator window	You	Dev	You modify the customizable elements of HTML templates to modify the appearance of Oracle Configurator window, or create new templates following documented structure.
Default image files for the background and UI controls of the Oracle Configurator window.	Oracle	Run	Image files appear in HTML templates and JavaScript controls.
Custom image files for the background and UI controls of the Oracle Configurator window.	You	Dev	You can customize the default set of files, or substitute your own files.
XML initialization message for the OC Servlet	You	Dev Run	You specify the XML parameters in the initialization message that your application passes to the runtime Oracle Configurator.
XML termination message from the OC Servlet	You	Dev Run	You parse the XML output from the UI Server's termination message, and pass data back to your host application.

Table 2–3 (Cont.) Required Elements and Actions for Oracle Configurator Custom Application

Element Required	Provided by...	Dev/Run	Action Required or Provided
Return URL Java servlet	You	Dev Run	You implement a servlet class that provides behavior after the Oracle Configurator window is used.
Oracle Configurator configuration engine	Oracle	Run	Automatically accessed by the CIO, through the OC Servlet.
Oracle Configurator Active Model	Oracle	Run	Processes Model structure and rules to create configurations.

Note: Detailed information on building a custom application is *not* provided in the Oracle Configurator documentation set.

Part II

Data

Part II presents information about working with the Oracle Configurator schema as described in Section 1.2, "Database Tasks" on page 1-2. Part II contains the following chapters:

- Chapter 3, "Database Instances"
- Chapter 4, "The CZ Schema"
- Chapter 5, "Populating the CZ Schema"
- Chapter 6, "Migrating Data"
- Chapter 7, "Synchronizing Data"
- Chapter 8, "CZ Schema Maintenance"

3

Database Instances

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:

- Database Uses
- Multiple Database Instances
- Development Database Instance
- Production Database Instance

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

3.1 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, BOMs, and saved configurations

Oracle supports using either one or two separate databases to fulfill all necessary operations during an Oracle Configurator implementation and deployment:

- One database instance for all uses
- Two separate database instances on one or two machines:
  - Development Database Instance
  - Production Database Instance

Using the Publication Mode applicability parameter (Production or Test) on Models in a single database instance prevents interference between production, test, and development versions of configuration models. See Section 17.2.3.1, "Mode" on page 17-9 for details.

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

For more information about implementing Oracle Configurator in two separate database instances, see Section 3.2, below.

3.2 Multiple Database Instances

Once a Configurator 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: Using more than two separate database instances for an Oracle Configurator implementation can cause unresolvable problems with data synchronization.

The operations that commonly involve a separate development and production database instance are:

- Migrating or importing data into the CZ schema
- Publishing configuration models to a production CZ schema
- System testing configuration models

When working with two database instances, the one into which you are logged in is usually referred to as *local* or current, while the other is *remote*. A database instance

that is remote and serves as a source or target of some data operation with the current database instance is also called a remote server.

3.2.1 Reasons for Multiple Database Instances

Although it is possible to implement and deploy Oracle Configurator using only one database instance, many projects use two database instances to separate development from production use:

- Development Database Instance, which can serve as:
  - Import target
  - Publication source and target
  - Migration target
  - BOM Synchronization source or target
- Production Database Instance, which can serve as:
  - Import source
  - Publication target
  - Migration source
  - BOM Synchronization source

See Section 1.2, "Database Tasks" on page 1-2 for scenarios using separate development and production database instances.

3.2.1.1 Import Source and Target

The imported data used to develop a runtime Oracle Configurator should be production data. The production database serves as the import source. In order to develop a BOM-based configuration model, BOM data must be imported into the Oracle Configurator schema. The Oracle Configurator schema serves as the import target. For information about data import, see Chapter 5, "Populating the CZ Schema".

3.2.1.2 Publication Source and Target

The configuration models must be published from the development database instance for availability to system testing or production in the same or a different database. The development database instance is the publication source. The system

test or production database is a publication target. For information about publishing, see Section 17.3, "Publishing a Configuration Model" on page 17-10.

If you change the publication source or target or use a cloned source or target, you will likely need to synchronize the publication data. See Chapter 7, "Synchronizing Data".

Decommissioning a target database instance does not cause synchronization problems. You can use the Model Publishing window in Oracle Configurator Developer to delete the existing publications on the target instance. See the *Oracle Configurator Developer User's Guide* for additional publishing information.

3.2.1.3 Migration Source and Target

In cases where you are moving your Configurator implementation or deployment from one database instance to another, you may need to migrate configuration model data. For more information about migration, see Chapter 6, "Migrating Data".

3.2.1.4 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 Chapter 7, "Synchronizing Data".

3.2.2 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 or target database and the local Oracle Configurator schema.

See Section A.4, "Server Administration" on page A-3 for general information on setting up database links. For details on running the concurrent programs, see Section B.2.1, "Define Remote Server" on page B-5, and Section B.2.2, "Enable Remote Server" on page B-6.

3.2.3 Instance and Host Machine Names

Multiple database instances can exist on a single or separate host machines. Both the database instance and the host machine have a name. The name of the database instance and host machine are relevant in all the uses listed in Section 3.2.1, "Reasons for Multiple Database Instances" on page 3-3.

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

Instance and host names are specified in various places for the correct operation of Oracle Configurator Developer, the CZ schema, and Configurator concurrent programs. The instance name for the CZ schema you use with Oracle Configurator Developer is identified in the `spx.ini` file. See Section 15.2, "Spx.ini" on page 15-3 for more information about the `spx.ini` file.

3.3 Development Database Instance

A development database instance is one in which you create your Oracle Configurator Model.

Note: There should only be one development database instance.

Note: Configuration models that will be available to end users should be *published* only from a single development environment.

Unit testing is implemented by using the **Test** button in Oracle Configurator Developer. This enables the developer to test a configuration model's rules and UI functionality in the development database instance, making sure periodic rule and

UI modifications work as desired. For additional information see the *Oracle Configurator Developer User's Guide*.

A system test environment is one in which you publish the Model in preparation for initial deployment, Configurator upgrades or new release, and Model updates. System testing tests:

- The performance of the configuration model
- End user access
- Security
- Integration customizations

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

If the system testing environment is within the development instance and not a separate database instance.

When you deploy a runtime Oracle Configurator and create a production Oracle Configurator database instance or publish configuration models, you may also create a maintenance Oracle Configurator database instance.

A maintenance environment is similar to a development environment as it is used to upgrade the CZ schema, refresh configuration data, fix and improve configuration models, and periodically republish the models. It is important to synchronize these 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 Chapter 7, "Synchronizing Data".

When you upgrade the release version of Oracle Configurator that your runtime Oracle Configurator runs against, you start by upgrading your Oracle Configurator schema. For information about updating your Oracle Configurator schema, see Section 8.2, "Refreshing or Updating the Production Schema" on page 8-1 and Section 8.4, "Redoing Sequences" on page 8-2. For information on upgrading from one release to another of Oracle Configurator, see the *Oracle Configurator Installation Guide*.

Once you have upgraded the Oracle Configurator schema, you must re-run the Generate Active Model command in Oracle Configurator Developer for each configuration model. There are precautions you must take so that you do not get

unpredictable behavior at run time. For example, overlapping applicability parameters for a publication results in unpredictable behavior. See Chapter 17, "Publishing Configuration Models".

3.4 Production Database Instance

A production environment is one in which end users of the runtime Oracle Configurator use the software in a production mode.

To prepare for deployment to your production environment, you should create a production-ready version of the Oracle Configurator schema that is populated with a published configuration model whose Publication Mode applicability parameter is set to Production.

If the development database and the production database are not on the same machine, then the server holding the production database must be defined and enabled. Enabling the remote server sets up the necessary database links between the development database and the production database.

Before you publish the configuration model, purging records flagged for deletion results in a more efficient use of machine resources. For more information about purging records, see Section 8.3, "Purging Configurator Tables" on page 8-2.

Possible deployment scenarios are:

- Client / Server (networked)
- Web

The Oracle Configurator schema is installed on the database server machine as part of your Oracle Applications database.

In a Web environment, the runtime Oracle Configurator user interface runs 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. For more information see Chapter 2, "Configurator Architecture".

The networked runtime Oracle Configurator can be integrated with those hosting applications that support Oracle Configurator. For a list of supporting applications refer to the *Oracle Configurator Release Notes*. The runtime Oracle Configurator could also be embedded in a custom Web application.

For information about publishing a configuration model to a production Oracle Configurator schema, see Chapter 17, "Publishing Configuration Models". For information about refreshing or updating a production Oracle Configurator schema, see Section 8.2, "Refreshing or Updating the Production Schema" on page 8-1.

4

The CZ Schema

This chapter provides a basic understanding of the Oracle Configurator schema, including information about:

- Import Tables
- Control Tables
- CZ_DB_SETTINGS Table

The Oracle Configurator schema is also called the CZ schema.

4.1 Characteristics of the Oracle Configurator Schema

For a description of the Oracle Configurator schema, see Section 2.1.3, "Oracle Configurator Schema" on page 2-4. See the *Oracle Configurator Release Notes* for a list of tables not in use.

4.1.1 Online Tables and Integration Tables

The CZ schema contains online tables 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 Section 4.1.2 for more information about the CZ subschemas.

The integration tables consist of import tables and control tables. See Section 4.2 for information about the import tables. See Section 4.3 for information about control

tables. See Chapter 5, "Populating the CZ Schema" for information about using the integration tables.

4.1.2 CZ Subschemas

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

ADMN - Administrative
CNFG - Configuration
GN - General Use
ITEM - Item-Master
LCE - Logic for Configuration (Active Model)
PB - Publication
PROJ - Project Structure
RP - Repository
RULE - Rule
UI - User Interface
XFR - Transfer specifications and control

See Appendix C, "CZ Subschemas" for a listing of tables in each subschema.

4.1.3 Database Links

The integration tables in the CZ schema are set with database links to the source database when the source database is on a different machine from the CZ schema.

4.1.4 Public Synonyms

The CZ schema does not use public synonyms.

4.1.5 Schema Customization

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

Various user expansion fields in the Oracle Configurator schema, such as USERNUM*n* and USERSTR*n* in the CZ_PS_NODES table, are available for custom use. The data in these user expansion fields is preserved during a schema upgrade or migration. For more information, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

4.2 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 and additional fields to manage the import and correlate the data with the existing data in the online table.

Import tables consist of:

- Online Data Fields
- Import Control Fields
- Surrogate Key Fields

Since 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 is named exactly as its online counterpart, but has a CZ_IMP prefix instead of just CZ_. For example, the imported data in CZ_IMP_ PROPERTIES populates CZ_PROPERTIES.

The import tables temporarily store extracted or legacy data that the concurrent programs access when creating, updating or deleting records in the CZ schema. The CZ_IMP tables are populated by the Populate or Refresh Configuration Models concurrent programs. For information about how data moves from sources outside the CZ schema through the import tables to the online tables, see Chapter 5, "Populating the CZ Schema".

For information about dependencies among import tables and import table codes, see Section 4.2.4, "Dependencies Among Import Tables" on page 4-5.

4.2.1 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. Table 4–1 describes the import control fields.

Table 4–1 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.
REC_STATUS	VARCHAR(4)	Output field that indicates the record's validation status. Dupl indicates that the record is a duplicate. Err indicates that the record has not been modified or inserted into the target database table because of an error in the transfer stage. F*nnn* indicates that the *nnn* field is an invalid foreign-key reference. N*nnn* indicates that the required *nnn* field has null data. Null indicates that the record status is open. Once this status is set, further processing of the record is suppressed. OK indicates that the data in this record now exists in the online database table. Pass indicates that the record is marked for either modification or insertion after the key resolution stage.
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.

4.2.2 Online Data Fields

Online 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.

4.2.3 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.

4.2.4 Dependencies Among Import Tables

Dependencies among import tables must be heeded especially when custom importing single tables. In Table 4–2, "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 or FSK_ITEMTYPE_1_EXT column in CZ_IMP_ ITEM_MASTER gets its value from CZ_IMP_ITEM_TYPE and helps in key resolution. FSK_ITEMTYPE_1_1 (default) or FSK_ITEMTYPE_1_EXT are populated depending on the indicator (0, 1, or 2) in CZ_XFR_TABLE.

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

A strong dependency means a value is required to successfully import that record. If "Default" is YES, there is already a default value in that column and import will succeed even if the dependency is strong and no value is imported. The following Table 4–2 lists the dependencies.

Table 4–2 Dependencies Among Oracle Configurator Schema Import Tables

Import Table Name	Depends on	for Foreign Surrogate Key	Type of dependencies	Default
CZ_IMP_DEVL_PROJECT	CZ_IMP_INTL_TEXT	FSK_INTLTEXT_1_1	STRONG	NO
CZ_IMP_INTL_TEXT	NO	NO	NO	NO
CZ_IMP_ITEM_MASTER	CZ_IMP_ITEM_TYPE	FSK_ITEM_TYPE	STRONG	YES
CZ_IMP_ITEM_ PROPERTY_VALUE	CZ_IMP_PROPERTY	FSK_PROPERTY_1_1	STRONG	NO
CZ_IMP_ITEM_ PROPERTY_VALUE	CZ_IMP_ITEM_ MASTER	FSK_ITEMMASTER_ 2_1	STRONG	NO
CZ_IMP_ITEM_TYPE	NO	NO	NO	NO
CZ_IMP_ITEM_TYPE_ PROPERTY	CZ_IMP_ITEM_TYPE	FSK_ITEMTYPE_1_1	STRONG	NO
CZ_IMP_ITEM_TYPE_ PROPERTY	CZ_IMP_PROPERTY	FSK_PROPERTY_2_1	STRONG	NO
CZ_IMP_PROPERTY	NO	NO	NO	NO
CZ_IMP_PS_NODE	CZ_IMP_INTL_TEXT	FSK_INTLTEXT_1_1	STRONG	NO
CZ_IMP_PS_NODE	CZ_IMP_ITEM_ MASTER	FSK_ITEMMASTER_ 2_1	STRONG	NO
CZ_IMP_PS_NODE	CZ_IMP_DEVL_ PROJECT	FSK_ DEVLPROJECT_5_ EXT	STRONG	NO

4.3 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 or Refresh Configuration Models concurrent programs, records in the CZ_XFR tables determine which import tables are enabled for import, what data is imported, and how the data is imported.

There are six tables that control the import process at the table and field level. The six control tables are:

- CZ_XFR_FIELDS
- CZ_XFR_PROJECT_BILLS
- CZ_XFR_RUN_INFOS
- CZ_XFR_RUN_RESULTS
- CZ_XFR_STATUS_CODES
- CZ_XFR_TABLES

CZ_XFR_TABLES identifies the mapping of the import table to the online table as well as the rules for importing the 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 for CZ_ITEM_MASTERS.DESC_TEXT inhibits the updating of the Item Master description. Example 4–1 shows how to set the field in the CZ_XFR_FIELDS table so that changes made to the BOM's Item description do not appear in Oracle Configurator Developer.

Example 4–1 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
```

4.4 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.

4.4.1 Editing the CZ_DB_SETTINGS Values

You can set the values that your installation requires by running the Modify Configurator Parameters concurrent program. See Section B.1.2, "Modify Configurator Parameters" on page B-3 for details on running the concurrent program.

4.4.2 Organization of the CZ_DB_SETTINGS Table

The parameters in the CZ_DB_SETTINGS table are organized into sections. The sections are identified in the SECTION_NAME field:

- IMPORT controls how BOM data is imported into the CZ schema
- LogicGen governs certain aspects of the way the Active Model l is generated from your Model
- ORAAPPS_INTEGRATE controls how Oracle Configurator integrates with other Oracle Applications.
- SCHEMA sets some general parameters that control the Oracle Configurator schema
- UISERVER governs certain aspects of the runtime Oracle Configurator user interface

Each section contains relevant database parameters and each parameter contains the following fields:

- DATA_TYPE specifies the datatype of the parameter. 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 setting default value is stored as string 10000, but Configurator interprets it as an integer.
- SETTING_ID is the database parameter identifier.
- VALUE is the default value that is seeded during an installation or upgrade of the database instance.

4.4.3 CZ_DB_SETTINGS Parameters

Table 4–3 lists the parameters that are seeded during an installation or upgrade of Oracle Configurator. You can modify the values of these parameters by running the Modify Server Definition concurrent program. For information on running concurrent programs, see Section A.1, "Running Configurator Concurrent

Programs" on page A-1. For specific information on modifying the parameters in the CZ_DB_SETTINGS table, see Section B.2.4, "Modify Server Definition" on page B-8.

Table 4–3 Settings in CZ_DB_SETTINGS Table

SETTING_ID	SECTION_ NAME	DATA_ TYPE	Default VALUE	More information in...
AltBatchValidateURL	ORAAPPS_ INTEGRATE	string	Null	Section 4.4.3.1
APPS_PREFER_UI_0	ORAAPPS_ INTEGRATE	string	671	Section 4.4.3.2
APPS_PREFER_UI_3	ORAAPPS_ INTEGRATE	string	521, 532, 660, 697, 702	Section 4.4.3.3
BadItemPropertyValue	IMPORT	CHAR (1)	F	Section 4.4.3.4
BatchSize	SCHEMA	string	10000	Section 4.4.3.5
BOM_REVISION	ORAAPPS_ INTEGRATE	string	Null	Section 4.4.3.6
CommitSize	IMPORT	integer	500	Section 4.4.3.7
DISPLAY_INSTANCE_ NAME	UISERVER	string	False	Section 4.4.3.8
FREEZE_REVISION	SCHEMA	string	System setting	Section 4.4.3.9
GenerateGatedCombo	LogicGen	string	YES	Section 4.4.3.10
GenStatisticsBOM	IMPORT	string	NO	Section 4.4.3.11
GenStatisticsCZ	IMPORT	string	NO	Section 4.4.3.12
MAJOR_VERSION	SCHEMA	integer	System setting	Section 4.4.3.13
MaximumErrors	IMPORT	integer	10000	Section 4.4.3.14
MINOR_VERSION	SCHEMA	string	System setting	Section 4.4.3.15
MULTISESSION	IMPORT	integer	0	Section 4.4.3.16
OracleSequenceIncr	SCHEMA	integer	20	Section 4.4.3.17
PsNodeName	ORAAPPS_ INTEGRATE	string	RefPartNbr	Section 4.4.3.18

Table 4–3 (Cont.) Settings in CZ_DB_SETTINGS Table

SETTING_ID	SECTION_ NAME	DATA_ TYPE	Default VALUE	More information in...
PublicationLogging	ORAAPPS_ INTEGRATE	YES / NO	NO	Section 4.4.3.19
PublishingCopyRules	ORAAPPS_ INTEGRATE	string	YES	Section 4.4.3.20.
RefPartNbr	ORAAPPS_ INTEGRATE	string	CONCATENATED _SEGMENTS	Section 4.4.3.21
ResolvePropertyDataType	ORAAPPS_ INTEGRATE	string	YES	Section 4.4.3.22
RestoredConfigDefaultMo delLookupDate	ORAAPPS_ INTEGRATE	string	config_creation_ date	Section 4.4.3.23
Revision Date / User	SCHEMA	any string		Section 4.4.3.24
RUN_BILL_EXPLODER	ORAAPPS_ INTEGRATE	string	YES	Section 4.4.3.25
SuppressSuccessMessage	UISERVER	YES / NO	NO	Section 4.4.3.26
UI_NODE_NAME_ CONCAT_CHARS	ORAAPPS_ INTEGRATE	string		Section 4.4.3.27
UseLocalTableInExtractio nViews	IMPORT	string	NO	Section 4.4.3.28
UtlHttpTransferTimeout	SCHEMA	integer	none	Section 4.4.3.29

4.4.3.1 AltBatchValidateURL

AltBatchValidateURL allows the batch validation process to bypass the URL that is normally used for batch validation. If Oracle Configurator uses Secured Sockets Layer (SSL), then this value must be specified. The value must be the non-secure URL.

To insert the AltBatchValidateURL into the CZ_DB_SETTINGS table, use the SQL*Plus statement shown in Example 4–2.

Example 4–2 Adding AltBatchValidateURL to CZ_DB_SETTINGS

```
INSERT INTO cz_db_settings (setting_id, section_name, data_type, valuem desc_
text) VALUES ('AltBatchValidateURL','ORAAPPS_
```

```
INTEGRATE',4,'http://servername.com:8808/configurator/oracle.apps.cz.servlet.UiS
ervlet','Non-secure URL')
```

4.4.3.2 APPS_PREFER_UI_0

APPS_PREFER_UI_0 indicates the host applications that invoke a Configurator DHTML user interface unless none is available, in which case they look for a Java applet user interface to invoke. The seeded value of APPS_PREFER_UI_0 is a comma-delimited list of host application IDs. The seeded value is:

- 671 - Oracle *i*Store

4.4.3.3 APPS_PREFER_UI_3

APPS_PREFER_UI_3 indicates the host applications that invoke a Configurator Java applet user interface unless none is available, in which case they look for a DHTML user interface to invoke. The seeded value of APPS_PREFER_UI_3 is a comma-delimited list of host application IDs. The seeded values are:

- 521 - Oracle TeleSales
- 532 - Oracle Sales for Communications
- 660 - Oracle Order Management
- 697 - Oracle Order Capture
- 702 - Bills of Material

4.4.3.4 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. Table 4–4 lists the valid values for BadItemPropertyValue setting and the disposition:

Table 4–4 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

Table 4–4 (Cont.) Valid Values for the BadItemPropertyValue Setting

Value	Disposition
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 will default to the property default value in the CZ_ITEM_PROPERTY_VALUES table.

4.4.3.5 BatchSize

BatchSize indicates the number of records that are modified before committing a transaction in batch operations such as import and logic generation.

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 this type of 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 because it is considerably more economical to commit many updates at once.

4.4.3.6 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 explosion procedure.

The value of BOM_REVISION is the Oracle Applications revision number. Valid values are "5.0.628" for Release 10.7, "11.0.28" for Release 11.0, and "11.5.0" for Release 11i. If the value is null (default), "11.5.0" is used. The call to the BOM explosion procedure checks up to the second decimal point of this value.

If the value is 11.5.*n*, then the 11*i* date format "YYYY-MM-DD" is used. Otherwise, the Release 10.7 or 11.0 date format "DD/MON/RR" is used.

4.4.3.7 CommitSize

CommitSize indicates the number of import records in each database transaction at a time, between commits.

4.4.3.8 DISPLAY_INSTANCE_NAME

DISPLAY_INSTANCE_NAME determines whether an **Instance Name** column appears in the Oracle Configurator Summary screen. 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 no components with multiple instances exist in the configuration, then the **Instance Name** column does not appear. If set to `False` but multiple instances exist in the configuration, then instance names appear in the **Description** column (instead of each item's description).

4.4.3.9 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 by the upgrade script.

4.4.3.10 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.

4.4.3.11 GenStatisticsBOM

GenStatisticsBOM set to Yes, forces the optimizer to update the internal statistics on the BOM_EXPLOSIONS table before running queries in our code. Generating statistics allows the optimizer to choose a better execution plan based on the current data structure in a table.

4.4.3.12 GenStatisticsCZ

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

4.4.3.13 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.

4.4.3.14 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.

4.4.3.15 MINOR_VERSION

MINOR_VERSION indicates the minor version label for the CZ schema. This value is read-only and is populated by the upgrade script.

4.4.3.16 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.
- A negative value means ignore other import sessions and run this import session immediately. Setting this parameter to a negative number is equivalent to disabling it.

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

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

4.4.3.17 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.

4.4.3.18 PsNodeName

PsNodeName indicates the source field to be loaded into the NAME field in the CZ_PS_NODES table in the CZ schema. This 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.

4.4.3.19 PublicationLogging

PublicationLogging indicates whether a trace of the publication process is logged into 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 Section A.6, "Viewing Log Files" on page A-4.

4.4.3.20 PublishingCopyRules

PublishingCopyRules indicates whether or not configuration rules are copied during publishing. If set to NO, then the Rule subschema of the Model being published is not copied to the production environment and the publishing process is faster.

Note: Setting 'PublishingCopyRules' to 'No' only affects you if Oracle Configurator incorporates changes to logic generation that is incompatible with previous versions of Oracle Configurator. If published models have no rules, then it won't be possible to generate logic for the published models. Customers using the No setting would need to republish all of their published models.

4.4.3.21 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 import process to construct BOM 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 Configurator Administrator must manually set RefPartNbr to 'SEGMENT1' by running the Modify Configurator Parameters concurrent process.

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 is 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 be longer than 1000 characters, the limit of the CZ_PS_NODES.NAME field. Any description longer than 1000 characters is truncated. The default separator is a dot (.). Other separators are '|', '-', or a custom value. See the *Oracle Inventory User's Guide* for more information about setting up part numbers.

When running either the Populate or Refresh Configuration Models concurrent program, you can enter multi-segment items in the From Item and To Item input fields. When entering multi-segment item names, you must include any separators that exist in the item's part number.

Warning: To update an existing Model in Configurator Developer to use multi- segment part numbers, you must either reimport or refresh the BOM.Confirm that the BOM is getting re-exploded during import. The DB_SETTINGS.RUN_BILL_ EXPLODER should be Yes.

4.4.3.22 ResolvePropertyDataType

ResolvePropertyDataType indicates whether Catalog Descriptive Elements are imported as Properties with a string or a number data type. If the value for this setting is NO, all Catalog Descriptive Elements are imported as Properties with a string data type. If the value of this setting is YES, a Catalog Descriptive Element is imported as a Property with a number data type only if all of its imported values can be interpreted as numbers; otherwise it is imported as a Property with a string data type.

4.4.3.23 RestoredConfigDefaultModelLookupDate

This setting controls which publication Oracle Configurator uses on an order when called from Order Management. If this setting exists, then Oracle Configurator uses the order line creation date. If this setting does not exist, Oracle Configurator uses a publication that is valid at the date and time when configurating an order.

For more information, see "DEFAULT_RESTORED_CFG_DATES" on page 18-41.

4.4.3.24 Revision Date and User

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

4.4.3.25 RUN_BILL_EXPLODER

RUN_BILL_EXPLODER indicates whether or not the import process invokes the BOM_EXPLODER procedure, which explodes the BOMs before the import process extracts data.

Note: The Populate or Refresh Configuration Models concurrent programs do not explode BOMs when importing from a remote server. See Section 5.2.6, "Exploding BOMs in Oracle Applications" on page 5-8 for details.

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 Chapter 5, "Populating the CZ Schema" for more information on exploding a BOM.

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, the procedure calls the BOM 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, the concurrent program transfers the BOMs that are flagged for import in the CZ_XFR_PROJECT_BILLS table without running the BOM exploder first.

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 BOMs exploded in the BOM_EXPLOSIONS table in the Oracle Applications database. Selected BOM Items come from the BOM_BILL_OF_MATERIAL and the BOM_INVENTORY_COMPONENTS tables.

4.4.3.26 SuppressSuccessMessage

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

NO - After fixing a validation error a runtime success message is displayed.

YES - After fixing a validation error a runtime success message is not displayed.

To insert SuppressSucessMessage into CZ_DB_SETTINGS, use the SQL*Plus statement shown in Example 4–3.

Example 4–3 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')
```

4.4.3.27 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 separating each text string is a comma surrounded by two spaces. (For example: "AT62431 , Sentinal Custom Laptop"). You can change the character Configurator Developer uses to separate each string by running the Modify Configurator Parameters concurrent process.

4.4.3.28 UseLocalTableInExtractionViews

When UseLocalTableInExtractionViews is set to Yes, definitions of some of the import extraction views include the DUAL table in the join. This setting is ignored if the import server is local.

4.4.3.29 UtlHttpTransferTimeout

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

To insert UtlHttpTransferTimeout into the CZ_DB_SETTINGS, use the SQL*Plus statement shown in

Example 4–4 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*.

5

Populating the CZ Schema

This chapter reviews the various means by which data is inserted into the CZ schema, with detailed explanations of data import:

- Standard Import
- Custom Import

For information about the CZ schema generally, see Chapter 4, "The CZ Schema".

5.1 Overview

This overview lists the kinds of data that populate the CZ schema during development and runtime use of an Oracle Configurator, and by what means that data is inserted.

5.1.1 Kinds of Data Stored in the CZ Schema

The data stored in the CZ schema includes:

- Configuration models:
  - Item and Model structure data
  - Configuration rules
  - User interface definitions
  - Publication records
- Configurations

See Section 5.1.2, "Means of Populating the Oracle Configurator Schema" on page 5-2 for information on how this data is inserted. See Section 5.2, "Standard

Import" on page 5-3 for more details about the specific kinds of Inventory and BOM data stored in the CZ schema.

5.1.2 Means of Populating the Oracle Configurator Schema

The CZ schema is populated with data through various means:

- Concurrent programs used to import item and Model structure data from outside sources into the schema (see Section B.4, "Populate and Refresh Configuration Models Concurrent Programs" on page B-11)
- Scripts or concurrent programs used to migrate item and Model structure data from outside sources into the schema (see Chapter 6, "Migrating Data")
- Direct submits from user actions in the Oracle Configurator Developer to insert configuration model data (see the *Oracle Configurator Developer User's Guide*)
- Functional Companions designed to populate CZ table fields with configuration data that cannot be inserted directly using runtime Oracle Configurator (see the *Oracle Configuration Interface Object (CIO) Developer's Guide*)
- Configuration attributes custom designed and attached to certain nodes of configuration models (see *Oracle Configurator Methodologies*)
- End user actions in the runtime Oracle Configurator to store saved configurations (see *Oracle Configuration Interface Object (CIO) Developer's Guide*)

Of these means of populating the Oracle Configurator schema, data import is the starting point of most implementations, and is the focus of this chapter. There are two types of data import:

- Standard import of Oracle Applications BOM and Inventory data into the CZ schema (see Section 5.2, "Standard Import" on page 5-3)
- Custom import of data not handled by a standard import (see Section 5.3, "Custom Import" on page 5-17)

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

5.1.3 Target Tables

For information about where various kinds of data are stored in the CZ schema, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

5.2 Standard Import

A standard import consists of transferring data from Oracle Applications Bills of Material (Releases 10.7, 11.0, or 11*i*) to Oracle Configurator Release 11*i*.

When developing a configuration model, Oracle Configurator Developer accesses the CZ schema, not the Oracle Applications Inventory and Bills of Material schemas. Likewise, during configuration, the runtime Oracle Configurator accesses the CZ schema. In order to develop a configuration model for creating configurations of BOMs that participate in downstream processes such as ordering, the CZ schema must contain representations of the BOM structure, rules, and item data that exactly match the data used to fulfill the order.

5.2.1 Inventory and BOM Data That Can Be Imported

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

- Bills of Material (BOM) structure (ATO and PTO BOM Models)
- Inventory data
- ATO or PTO BOM 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.

5.2.2 Overall Standard Import Procedure

The overall procedure for a standard import is:

1. Determine the import source and target (see Chapter 3, "Database Instances")
2. Prepare the data (see Section 5.2.4 on page 5-4).
3. Define and enable the source server for import, if remote (see Section 5.2.5 on page 5-7).
4. Explode the BOM Models you want to import, if the import source is remote (see Section 5.2.6 on page 5-8).

5. Optionally identify specific data to be ignored during the import (see Section 5.2.7 on page 5-9).
6. Run the Populate Configuration Models concurrent program to import data (see Section 5.2.8 on page 5-12).

   Import lowest level submodels before importing higher level and root models to diminish duplication of data in the CZ schema.
7. Verify that the data import succeeded (see Section 5.2.9 on page 5-14).
8. If you re-import the same BOM from a different source, you must first synchronize your BOM-based configuration models with that new source (see Chapter 7, "Synchronizing Data" on page 7-1).
9. Because repeated data imports can result in large amounts of logically-deleted items in the CZ schema, run the Purge Configurator Tables concurrent program to improve database performance. For more information, see Section B.1.3, "Purge Configurator Tables" on page B-4.

5.2.3 Determining Import Source and Target

The source of imported data is also called the import source or import server. The import source should be a production database. Oracle Configurator supports importing BOM 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 Section 5.2.5, "Defining and Enabling a Server for Import" on page 5-7 for information about changing the import server.

The target of the imported data is the database instance you have designated for developing your BOM-based configuration model. The target is the database in which you run the import.

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

5.2.4 Preparing the Data

For purposes of consistency with other processes in your business, use production data. Preparing the data for 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.

Determine which version of Oracle Applications you are importing from. You can only import BOMs from Release 10.7, 11.0 and 11*i* to Release 11*i*. Standard import requires that BOMs be complete and identified at the desired root. If a BOM is not complete, the populate concurrent program displays a warning message.

To create a BOM Model in Oracle Applications, you must first define the Items (see Section 5.2.4.1) and then their hierarchical relationship in a BOM Model (see Section 5.2.4.2).

5.2.4.1 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 items before importing data to the CZ schema. See Chapter 14, "Multiple Language Support" on page 14-1.

In Oracle Applications Inventory:

- Define the Items of your BOM 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 or can be used to create a BOM.
- 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).
- Specify the **OM Indivisible** check box to determine whether Item quantities should be treated as decimals or integers (see Section 5.2.7.6 on page 5-11).

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. A BOM Option Class typically contains one or more Standard Items.

BOM Standard Items are Inventory Items that are related in some way. For example, an Option Class called Speaker might contain the Standard Items Woofer, Tweeter, and Cabinet. A Standard Item can be a parent of components or can be a component of one or more parent items. Purchased items, subassemblies, and

finished products can be Standard Items. Standard Items are never planning items, Option Classes, or Models. See Section 5.2.7.6 on page 5-11 for details about importing Standard Item quantities as integers or decimals.

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

Each **Descriptive Element** is imported as a Property and is associated with the corresponding Item Type imported from the Catalog Group. Imported items associated with the Item type contain the associated Descriptive Element value as a Property value.

By default, all Descriptive Element values are imported into Oracle Configurator Developer as text. However, you can modify how these values are imported using the ResolvePropertyDataType setting in the CZ_DB_SETTINGS table. For details, see Section 4.4.3.22, "ResolvePropertyDataType" on page 4-17.

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*.

5.2.4.2 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.
- 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.

- 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.

For more information about the **Check ATP** and **ATP Components** settings, see the *Advanced Supply Chain Planning and Global Order Promising Users Guide.*

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, see the *Oracle Bills of Material User's Guide*.

5.2.5 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 Section A.4, "Server Administration" on page A-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 concurrent request to disable the local server for import, and then define and enable the remote server where the import source data is stored. To run this concurrent program, see Section B.2.4, "Modify Server Definition" on page B-8.

Oracle recommends that you define only one server for import. If an import server is changed after a BOM has been imported, then the configuration models must be synchronized to the BOMs on the new import server. For details on synchronizing the configuration models with the BOMs on the newly defined remote server, see Section 7.2.1, "The BOM Synchronization Process" on page 7-2.

5.2.6 Exploding BOMs in Oracle Applications

If you are importing from Bills of Material (Releases 10.7, 11.0, or 11*i*) in another instance (remote server), you must explode the BOM Model prior to submitting the concurrent request to import the BOM Models into the CZ schema.

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

5.2.6.1 Exploding a BOM in Release 11*i*

To explode a BOM Model in Oracle Applications, Release 11*i*:

1. Log into Oracle Applications using the appropriate username and password.
2. Select the Order Management responsibility.
3. Select **Orders, Returns > Sales Orders**.
4. Enter all required information in the **Main** tabbed region.
5. Click on the **Line Items** tabbed region.
6. Select the item that you want to import into Oracle Configurator from the list of values. This is the same model that you select when running the Populate Configuration Models concurrent program.
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 window.
9. Repeat steps 1 through 10 for each BOM Model that you want to import into the CZ schema.

5.2.6.2 Exploding a BOM in Release 10.7 or 11.0

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

1. Log into Oracle Applications using the appropriate username and password.
2. Select the Order Entry responsibility.
3. Navigate to the Sales Orders window, 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 exactly the same model that you will select from the list of values when you run the Oracle Configurator Populate Configuration Models concurrent program.

5. Enter 1 in the **Qty** field, then click **Configurator**.
6. After all the BOM Model's components are displayed, **Cancel** to close the Configurator window.
7. Repeat steps 1 through 7 for each BOM Model that you want to import into the CZ schema.

5.2.7 Controlling the Data

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. There are six tables that control the import process. For more information about the control tables, see Section 4.3, "Control Tables" on page 4-6. See Section 5.2.8, "Importing the Data" on page 5-12 for information about identifying what gets imported.

5.2.7.1 Importing Data Into Specific Tables

When you import data, you must be aware of the dependencies between the import tables. For more information, see Table 4–2, " Dependencies Among Oracle Configurator Schema Import Tables" on page 4-6.

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.

For general information on running concurrent programs, see Section A.1, "Running Configurator Concurrent Programs" on page A-1. For details on importing data into specific tables, see Section B.9.1, "Importing Data into Specific Tables" on page B-27.

You can also display the current tables to be imported by selecting the concurrent program, Show Tables To Be Imported. For more information, see Section B.9.2, "Show Tables to be Imported" on page B-28.

5.2.7.2 Importing Data from Specific Fields

You can customize which fields in the tables listed in CZ_XFR_TABLES are extracted and imported. See the Configurator eTRM on Metalink, Oracle's technical support Web site 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*Plus.

5.2.7.3 Populating Import Tables

The import tables below are listed in the order in which the concurrent programs and SQL* Plus import procedures populate them:

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

5.2.7.4 Modifying EXPLOSION_TYPE

You can modify the CZ_XFR_PROJECT_BILLS.EXPLOSION_TYPE field for previously imported bills to indicate how the BOM exploder should handle standard items. The possible values for this field are OPTIONAL (default), ALL, or INCLUDED. See the Configurator eTRM on Metalink, Oracle's technical support Web site for more information about CZ_XFR_PROJECT_BILLS and other control tables.

There is no concurrent program to complete this customization. Modification of the EXPLOSION_TYPE field can only be accomplished by using SQL*Plus.

5.2.7.5 Identifying a BOM for Import

CZ_XFR_PROJECT_BILLS table identifies a top-level item from a bill of material (BOM) in Oracle Applications for import into the CZ schema. Every imported bill must be represented in CZ_XFR_PROJECT_BILLS.

The TOP_ITEM_ID and ORGANIZATION_ID for each imported BOM 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 BOMs 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 items in Oracle Applications.

For detailed information about the control tables, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

5.2.7.6 Importing Decimal or Integer Quantities

You can specify whether Items are imported as integers or decimals using the profile option CZ: Populate Decimal Quantity Flags. This 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 will be set to 1, which means that decimal quantities are allowed.
  - If INDIVISIBLE_FLAG is 1, then DECIMAL_QTY_FLAG in both tables will be set to 0, which means that decimal quantities are not allowed.

If you change the profile option from `No` to `Yes`, you must refresh all existing Models so they will 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: All Oracle Applications that are integrated with Oracle Configurator do not support decimal quantities for BOM Standard Items. Additionally, Oracle Configurator offers limited support for using decimal quantities. See specific product documentation and Metalink for current information on how specific modules support 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 Configuration Interface Object (CIO) Developer's Guide*.

5.2.8 Importing the Data

The Populate and Refresh Configuration Models concurrent programs in Oracle Applications are designed to perform the data import. These concurrent programs import BOM structure (ATO, PTO Models, structure and rules) and require that the BOMs be complete and identified at the desired root (see Section B.4, "Populate and Refresh Configuration Models Concurrent Programs" on page B-11).

Data import can be customized to run or suppress the transfer of some of this data (see Section 5.2.7, "Controlling the Data" on page 5-9).

If you are not importing from the same remote (import) server from which you originally imported the BOM Models, synchronize your BOM-based configuration models with the BOM Models on the new import server (see Chapter 7, "Synchronizing Data" on 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.

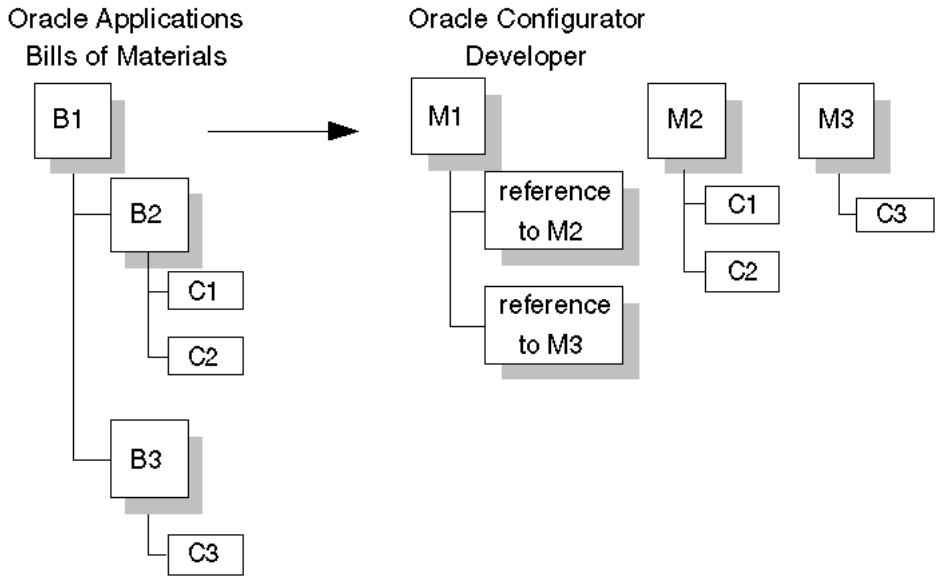
5.2.8.1 BOM Model With Child BOM Models

This section presents the specific results in Oracle Configurator Developer when initially importing BOM Models with child BOM Models.

When an ATO or PTO BOM Model is imported, a Model node is created in Oracle Configurator Developer. If the imported BOM Model has components that are ATO or PTO Models, then the import process creates Model structure for each child Model and a corresponding Reference node in the parent Model.

For example, if you have a BOM (B1) that contains two child BOMs (B2 and B3), then importing B1 results in three corresponding Models (M1, M2, and M3) in the Configurator Developer Repository. Since B2 and B3 have child nodes, M2 and M3 have corresponding child nodes. M2 contains child nodes C1 and C2, and M3 contains child node C3. Figure 5–1 illustrates this result in Oracle Configurator Developer.

Figure 5–1 Initial Import of BOM Model with Submodels



See Section 5.2.10, "Refreshing Imported Data" on page 5-15 for limitations when refreshing BOM Models that have references.

5.2.8.2 BOM Model with a Common Bill

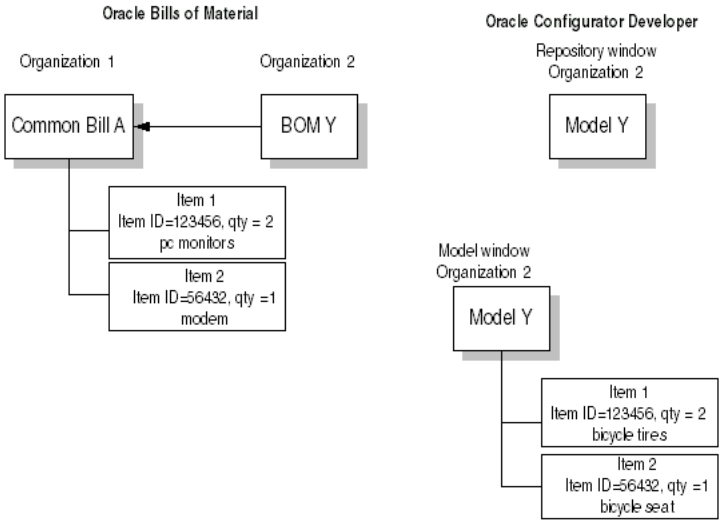
This section presents the specific results in Oracle Configurator Developer when initially importing BOM Models with a Common Bill.

A common bill is a read-only shared bill defined in the OM Validation Organization that can be pointed to when defining bills in other organizations. For more information about common bills, see the *Oracle Bills of Material User's Guide.*

When a BOM that points to a common bill is imported into the CZ schema, the imported BOM Model is visible in the Oracle Configurator Developer Repository

window, but the common bill is not. When the imported BOM Model is opened in the Oracle Configurator Developer Model window, the common bill's components are visible and available with no indication that they are from a common bill. Figure 5–2 shows a BOM Model Y in Organization 2 pointing to a common bill in Organization 1. The common bill has 2 items: item id 123456 is a pc monitor and has a quantity of 2 and item id 56432 is a modem and a quantity of 1. When BOM Model Y is imported into the CZ schema, it appears in the OCD Repository window. Common Bill A does not appear in the Repository window. When Model Y appears in the OCD Model window, items 123456 and 56432 also appear as child nodes of Model Y. Item 123456 in Organization 2 is a bicycle tire and item 56432 is a bicycle seat. The actual items are different, but the item ID and the quantity are the same as what was defined in common bill A.

Figure 5–2 BOM Model Pointing to a Common BIll



5.2.9 Verifying the Data Import

After you import data into the CZ schema, start Oracle Configurator Developer to view the Item Master and Model(s) containing the imported data. All Items imported into the CZ schema are displayed in the Oracle Configurator Developer Item Master. Items imported into the CZ schema Item Master can be edited.

The disposition of the import can be determined by examining the DISPOSITION field in the CZ_IMP tables. For more information about the DISPOSITION field see Table 4–1, " Import Control Fields" on page 4-4.

5.2.10 Refreshing Imported Data

Regardless of which source you use to import data into the CZ schema, you will occasionally need to refresh the schema with changes and updates for enterprise-wide consistency. Published configuration models maintain all relationships associated with the refreshed data to minimize Model maintenance when data is modified.

After you refresh a BOM Model, all changes made in Oracle Bills of Material are reflected in Oracle Configurator Developer.

This sections presents the specific results in Configurator Developer when refreshing BOMs in which the following changes have been:

- Changed BOM Model Type
- Changed BOM Model References
- New BOM Model With References to Already Imported BOM Models

5.2.10.1 Changed BOM Model Type

When you refresh a BOM Model that has changed from a PTO to an ATO Model in Oracle Bills of Material, the Refresh concurrent program (either Refresh A Single Configuration Model or Refresh All Configuration Models) checks to see if the Model supports multiple instantiation in Configurator Developer. For more information about multiple instantiation, see the *Oracle Configurator Developer User's Guide*.

If the Model in Oracle Configurator Developer supports multiple instantiation, then the concurrent program displays a message indicating that the BOM Model cannot be an ATO Model. When this occurs, the concurrent program fails and does not update the BOM Model information in Configurator Developer.

If the Model in Oracle Configurator Developer does *not* support multiple instantiation, then the concurrent program updates the Model and changes it to an ATO Model.

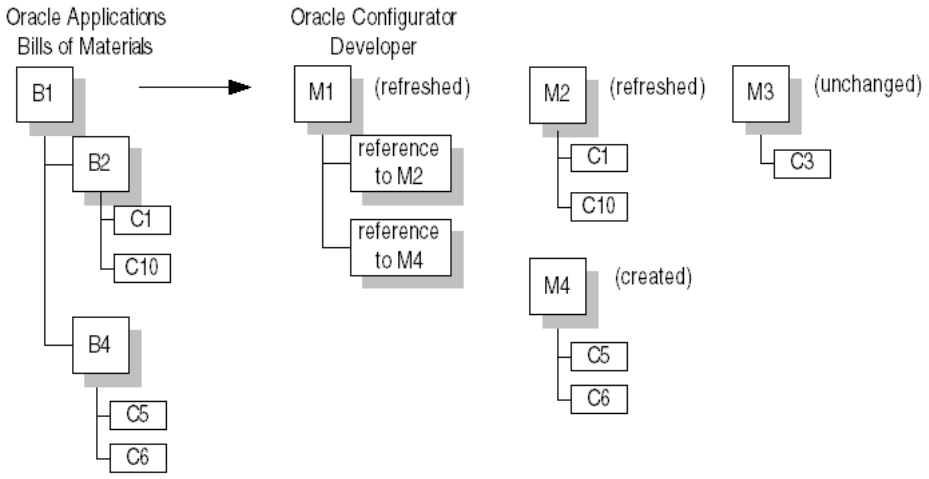
For more information about multiple instantiation, see the *Oracle Configurator Developer User's Guide.*

5.2.10.2 Changed BOM Model References

Replacing one child BOM 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, and remains in the Configurator Developer Repository.

Using the example presented in Figure 5–1, you modify B1 in Bills of Material so that it no longer references B3, but now references 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 B1 by running either the Populate Configuration Models or Refresh a Single Configuration Model concurrent program, you see the corresponding Models M1 and M2 refreshed in Oracle Configurator Developer. M4 is created corresponding to B4 and M3 remains unchanged. Figure 5–3 illustrates this result in Oracle Configurator Developer.

Figure 5–3 Populate and Refresh Modified BOM Model



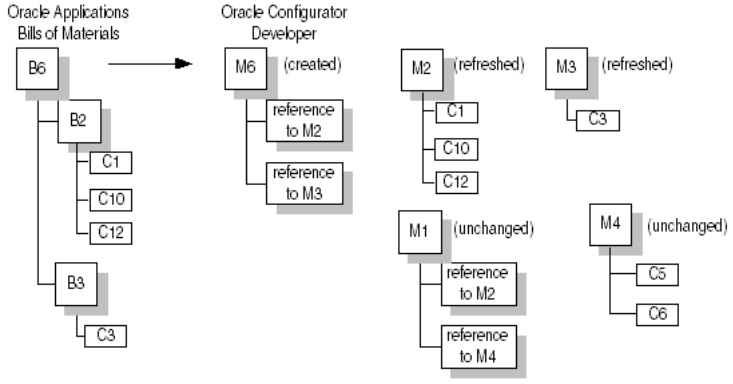
5.2.10.3 New BOM Model With References to Already Imported BOM Models

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

Using the example presented in Figure 5–2, you create BOM Model B6 in Oracle Bills of Material. B6 references B2 and B3. When you import B6 by running the Populate Configuration Models concurrent program, you see a new corresponding Model M6 in Oracle Configurator Developer, and updated versions of M2 and M3.

M1 now references the updated M2. Figure 5–4 illustrates this result in Oracle Configurator Developer.

Figure 5–4 Import a New BOM Model with References to Existing BOM Models



M1 and M6 both reference M2. When B6 is imported into the CZ Schema, M2 is refreshed with a new child node C12. M1 was not refreshed. Importing M6 might create problems for M1 because the logic and UI may no longer be valid with the changes and updates. The Oracle Configurator Developer user should regenerate the Active Model and UI for M1.

If M1 was published prior to the refreshing of M2, the runtime Oracle Configurator end user can still use M1 that references the original M2, as well as the publication of M6 that references the refreshed M2, because the publishing process creates a copy of the configuration model at the time of publication. For more information on publishing see the *Oracle Configurator Developer User's Guide.*

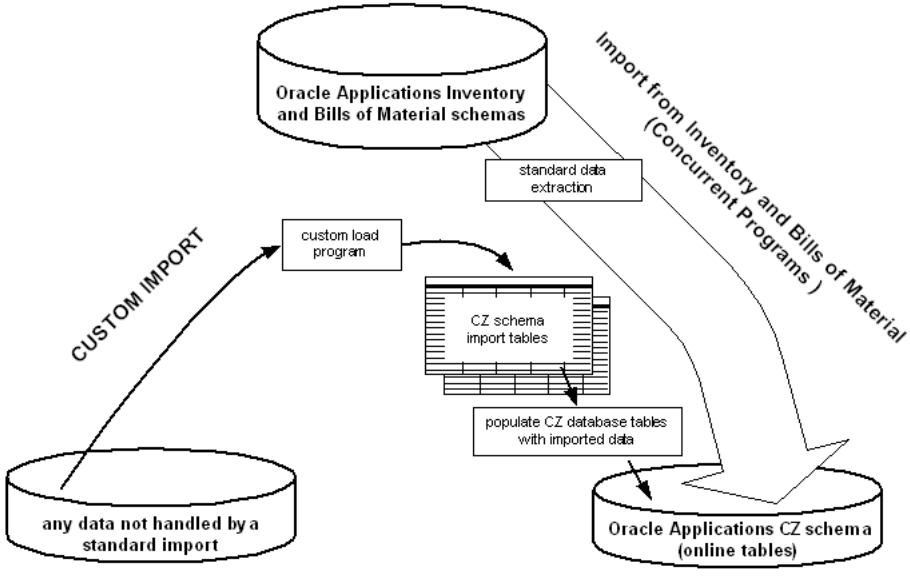
5.3 Custom Import

A custom import is required for importing data not handled by a standard import, including legacy data from non-Oracle Applications databases. See Section 5.2, "Standard Import" on page 5-3 to determine whether your data requires a custom data import.

5.3.1 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 concurrent programs, a custom import requires custom extraction, transfer, and load into the import tables. Figure 5–5 on page 5-18 shows where in the process the two kinds of data import are different.

Figure 5–5 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 run to populate the online tables with the extracted data used by the runtime Oracle Configurator. 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 Section 5.3.3, "Custom Import Procedure" on page 5-20 for information about performing a custom import.

After successfully importing any legacy data needed for modeling 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 using the Test button. See the *Oracle Configurator Developer User's Guide* for more information.

5.3.2 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

In order 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 Configurator eTRM on Metalink, Oracle's technical support Web site, for detailed information about all import table fields. See also Table 4–2, " Dependencies Among Oracle Configurator Schema Import Tables", on page 4-6 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 Section 5.2.7, "Controlling the Data" on page 5-9 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 DHTML Summary screen. For more information about the DHTML window and Summary screen see the *Oracle Configurator Developer User's Guide*.

5.3.3 Custom Import Procedure

If you are importing data not handled by a standard import, you must:

1. Identify and cleanse data for import.
2. Create and run custom extraction programs for the data you want to import:
   a. Write queries to extract the data into the required data transfer file format required by the import tables.
   b. Optionally create an ASCII file in that data transfer (DAT) format (see Section 5.3.4 on page 5-20).
   c. 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 Section 5.2.7 on page 5-9).
4. Run the concurrent program Populate Configuration Models to populate the CZ schema online tables with imported data from the import tables. See Section B.4 on page B-11.
5. Verify your import as described in Section 5.2.9 on page 5-14.

5.3.4 Required ASCII File Format for Custom Import

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

Example 5–1 shows a data transfer file that imports item types. Item type populates the third column of the 21-column import table CZ_IMP_ITEM_MASTER.

Example 5–1 Data Transfer File Format

```
||Memory Board|||||||||||||||||||
||Dual CPU|||||||||||||||||||
||Country|||||||||||||||||||
||System Console|||||||||||||||||||
||Server Console|||||||||||||||||||
||Disk Drive|||||||||||||||||||
||Storage Media|||||||||||||||||||
||Server Size|||||||||||||||||||
||Power Supply|||||||||||||||||||
```

```
||Matrix Printer|||||||||||||||||||
||SCSI Disk Drive|||||||||||||||||||
||Cache Memory|||||||||||||||||||
||Disk Array Model|||||||||||||||||||
||SCSI Type|||||||||||||||||||
||SCSI Cable|||||||||||||||||||
||SCSI Chaining|||||||||||||||||||
||SCSI Cabling Configuration|||||||||||||||||||
||Server Type|||||||||||||||||||
||System Size|||||||||||||||||||
```

6

Migrating Data

This chapter describes data migration to an 11*i* CZ schema:

- Migrating Data from Oracle SellingPoint to 11i
- Migrating Data from Another Oracle Configurator Schema

In this chapter, the term CZ schema identifies the schema used by Oracle Configurator, while the Oracle SellingPoint schema is the standalone schema used by Oracle SellingPoint Configurator.

6.1 Overview

Migration is the process of transferring data from one schema to another.

In order to migrate data from the Oracle SellingPoint schema to the Oracle Configurator schema of the Oracle Applications Release 11*i* database, or from one 11*i* database instance to another, the schema versions in the migration source and target must be the same.

Migration should only be run against an empty 11*i* target database containing the Oracle Configurator schema.

The CZ schema for Oracle Applications Release 11*i*, version 11.5.5 or higher is 15*i*. If your Oracle Applications installation uses schema 15*i* or higher and you need to migrate from Oracle SellingPoint, contact Oracle Support for more information.

Migration does not:

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

Because there is typically a large amount of data and migration time associated with saved configurations, migrating saved configurations is not recommended.

Warning: Data migration is a "one-time" process. Once migration is complete, do not repeat the process nor use the migration scripts to refresh data in the Oracle Applications database. Migration scripts are run once to bring the schema up to 11*i*.

6.2 Migrating Data from Oracle SellingPoint to 11*i*

The Oracle SellingPoint schema must be at 14e before beginning the migration process. The migration process will fail if the Oracle SellingPoint and Oracle Applications schemas are not the same version.

Warning: No data processing is allowed during data migration. Therefore, before migrating to Release 11*i* be sure you have met all of the prerequisites.

6.2.1 Prerequisites for Migrating Data from Oracle SellingPoint

The prerequisites for migrating data from Oracle SellingPoint to Oracle Configurator are:

- Determine the location of your source (Oracle SellingPoint) and target (Oracle Applications 11*i*) databases.
- Determine the SID, hostname, and listener port number of the source database.
- Verify that both the source Oracle SellingPoint schema and the target Oracle Configurator 11*i* database schema versions are 14e. See Appendix A.3, "Verifying Configurator Schema Version" on page A-3 for information about checking the major and minor version of a schema.

  If your Oracle SellingPoint schema is not 14e, see Section 6.2.2, "Upgrading Oracle SellingPoint to Schema 14e." on page 6-3.

  If your target Oracle Configurator 11*i* database schema version is not 14e, contact Oracle Support Services. Oracle Applications 11.5.1A, 11.5.2, 11.5.3, and 11.5.4 each contain a different version of Oracle Configurator, but are all CZ schema version 14e.

- Optionally, run the PURGE utility on your standalone source schema instance if you do not want to migrate logically deleted data. See the documentation for your current standalone version of Oracle SellingPoint for information about the PURGE utility.
- Set up the source 14e Oracle SellingPoint schema with the necessary migration packages and database link to the target 11*i* Oracle Applications database. See Section 6.2.3, "Setting Up Oracle SellingPoint Schema 14e for Migration" on page 6-4.
- Be sure the target Oracle Configurator 11*i* database schema is empty and not populated.
- Be sure no implementors are logged into Configurator Developer connected to the migration source or target database instances.
- Be sure no end users are connected to the migration source or target database instances, including production deployments, test deployments of the Oracle SellingPoint application, or a test runtime Oracle Configurator.

Once these prerequisites have been met, you can proceed with the data migration. See Section 6.2.5, "Running Data Migration From Oracle SellingPoint to Oracle Configurator" on page 6-7

6.2.2 Upgrading Oracle SellingPoint to Schema 14e.

In order to migrate an Oracle SellingPoint schema, you should so the following:

1. If your Oracle SellingPoint schema is not at 14e, then you must download patch 1847119 from Metalink. Patch 1847119 is a password protected patch for Oracle Configurator Developer Version 14.70. Contact Oracle Support Services for the information needed in order to download the patch.

   If your Oracle SellingPoint schema is at 14e, then skip to step 1 on page 6-4.

2. Unzip the patch into any folder.
3. As the Oracle Configurator user CZ, use SQL*Plus to connect to your Oracle Applications 11*i* schema (`SQL> connect cz/cz@appssid`). The CZ user has RESOURCE privileges. See Section A.2, "Connecting to a Database Instance" on page A-2 for further guidance.
4. In SQL*Plus, point to the DBAdmin/Server folder under the directory where you extracted the files (step 2) from the patch.

   For example:

a. Start SQL*Plus

b. Choose **File > Open**

c. Navigate to the `/DBAdmin/Migration` directory

d. Click Cancel (although the SQL prompt does not change, you are now in the `/DBAdmin/Migration` directory)

5. Run the UPGRADE_SERVER.sql script from the DBAdmin/Server directory. This script upgrades the SellingPoint schema to 14e and creates the UPGRADE_SERVER.log file.

   For example:

   ```
   SQL> @upgrade_server
   ```

   (Do not run Oracle Configurator SQL*Plus scripts from SQL Worksheet.)

6. Check the UPGRADE_SERVER.log file from step 5 for any errors or warnings.

7. Optionally verify the version of the upgraded schema. See Section A.3, "Verifying Configurator Schema Version" on page A-3 for details.

To proceed to migrating data, first set up the 14e Oracle SellingPoint schema with migration packages and a database link to the target Oracle Configurator schema (see Section 6.2.3).

6.2.3 Setting Up Oracle SellingPoint Schema 14e for Migration

Before setting up the Oracle SellingPoint schema for data migration to Oracle Configurator Release 11*i*, be sure the CZ_DB_SETTINGS parameter MAJOR_VERSION = 14 and MINOR_VERSION = e. See Section 6.2.2, "Upgrading Oracle SellingPoint to Schema 14e." on page 6-3.

1. Connect to the Release 11*i* Oracle Applications database containing your target 14e Oracle Configurator schema as user CZ. See step 3 in Section 6.2.2 on page 6-3.

2. In SQL*Plus, point to the DBAdmin/Server folder on the Oracle Configurator Developer CD-ROM or under the directory where Configurator Developer is installed. See step 4 in Section 6.2.2 on page 6-3.

Note: Be sure to have the most recent version of Oracle Configurator Developer CD-ROM.

---

3. Run the CZ_MIGRATE_SETUP.sql script from the DBAdmin/Server directory. See Section 6.2.4, "Output from CZ_MIGRATE_SETUP.SQL" on page 6-5 for information about what this script does.

   For example:

   ```
   SQL> @cz_migrate_setup
   ```

   (Do not run Oracle Configurator SQL*Plus scripts from SQL Worksheet.)

4. Enter the username for the source (standalone) database schema.
5. Enter the password for the source (standalone) database schema.
6. Enter the name of the database link to be created. The migration script uses the database link to move data between the source and target schemas. If GLOBAL_NAMES is set to true, then the link name must match the global name of the database to which you are creating the link. Otherwise, you can enter any name you want. See your database administrator for more information.
7. If the source (standalone) schema is located on the same database instance as the target (11*i*) schema, enter the TNS service name to access the source schema, then press **Enter** for the hostname and listener port number prompts.

   If the source schema is *not* located on the same database instance as the target schema, enter the SID, hostname, and the listener port number for the remote database where the source schema is located.

6.2.4 Output from CZ_MIGRATE_SETUP.SQL

The CZ_MIGRATE_SETUP.sql script in the DBAdmin\Migration folder creates migration packages, a database link, and the CZ_MIGRATE_SETUP.log file that contains errors and warnings. The following circumstances on the target cause an error or warning:

- The specified source database instance name does not have an associated database link
- The associated link is not functional
- There was a database error while populating the control table
- There was an unexpected database value setting
- The schemas on the source and target databases are not the same
- There are differences in table structure

The table comparison verifies whether:

- Like columns exist
- Data types match
- Nullable fields match

Note that mismatch messages appear during the table comparison due to the following known schema differences:

- The CZ_DB_SETTINGS table contains two additional records, DBLinkName and MigrationStatus, in the 11*i* schema. These fields are used for migration purposes.
- CZ_XFR_ TABLES contains many additional records in the 11*i* schema which are used for migration purposes.
- CZ_DB_LOGS is not migrated.
- The CHAR data type used in the standalone schema is replaced with the VARCHAR2 data type in the 11*i* schema.
- The CZ_EXP_TMP_LINES table exists in the 11*i* schema only.
- All CZ_IMP_xx (import) tables may exist only in the 11*i* schema.
- The PROPERTY_VALUE_NUM column of the CZ_PSNODE_PROPCOMPAT_ GENS table exists in the 4.2.2 schema only.
- All nullable fields in the standalone tables CZ_PRICING_STRUCTURE, CZ_ PSNODE_PROPCOMPAT_GENS, and CZ_RULE_FOLDERS are *not* nullable in the 11*i* schema.
- The CZ_END_USERS table is migrated, but the users are not defined as Oracle Applications users. To define end users as Oracle Applications users, see *Installing Oracle Applications*.

All discrepancies between the source and the target schemas must be fixed prior to migrating your data.

Note: A truncation error may occur during migration if a column data type is the same in both schemas but the length is different.

6.2.5 Running Data Migration From Oracle SellingPoint to Oracle Configurator

Be sure all prerequisites have been met before migrating data. SeSection 6.2.1, "Prerequisites for Migrating Data from Oracle SellingPoint" on page 6-2

1. As the Oracle Configurator user CZ, use SQL*Plus to connect to your Oracle Applications 11*i* target schema `SQL> connect cz/cz@appssid`). The CZ user has RESOURCE privileges. See Section A.2, "Connecting to a Database Instance" on page A-2 for further guidance.
2. On the target instance, run `CZ_RUN_MIGRATE.sql` from the DBAdmin\Migration folder to migrate your Oracle SellingPoint data to the target Oracle Configurator schema in the Oracle Applications Release 11*i* database.

   For example:

   ```
   SQL> @cz_run_migrate
   ```

   This script calls the MIGRATE procedure from the CZ_MIGRATE package. All errors and warnings are logged in the CZ_RUN_MIGRATE log file. The script copies all data from the source schema into the target schema and displays a status message when the migration is complete. If migration does not complete successfully, fix the reported issue and then rerun migration. For example, there may be a rollback segment problem that resulted in an unsuccessful migration.

   Note: The migration script uses queries such as "Insert into xx as Select * from yy" to copy data for most tables. Therefore, a rollback segment error can occur for large volume tables if the rollback segment for your database is set too low. The appropriate setting for the rollback segment depends on the size of your database and the volume of data it contains. If this occurs, then increase the size of your rollback segment and re-run `CZ_RUN_MIGRATE.sql`.

3. Upgrade to the latest version of Oracle Configurator 11*i* by applying the latest Configurator patch. Note that all Oracle Configurator patches are cumulative. For more information see the *Oracle Configurator Installation Guide.*
4. Download the corresponding Oracle Configurator Developer patch.

Do not run `CZ_RUN_MIGRATE.sql` again on the same target schema.

6.3 Migrating Data from Another Oracle Configurator Schema

In order to migrate CZ data from one Oracle Configurator 11*i* instance to another Oracle Configurator 11*i* instance you must be using Oracle Configurator version 11.5.7.17.44 or higher.

To migrate an Oracle Configurator 11*i* schema, do the following:

1. Check the version of both the Oracle Configurator 11*i* source and target database schemas.

   Both the source and the target must be at the same level. 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 level.

   See Section A.3, "Verifying Configurator Schema Version" on page A-3 for details.

2. Be sure no implementors are logged into Configurator Developer connected to the migration source or target database instances.

3. Be sure no end users are connected to the migration source or target database instances, including production deployments, test deployments of the Oracle SellingPoint application, or a test runtime Oracle Configurator.

4. On the source Oracle Configurator schema, delete any Models from the Oracle Configurator Developer Repository that do not have to be migrated into the target database schema.

5. Run the Purge Configurator Tables concurrent program to clean up the source schema prior to migrating the data. For more information see Section B.1.3, "Purge Configurator Tables" on page B-4.

6. Be sure the target Oracle Configurator schema is empty. On the target database run the Setup Configurator Data Migration concurrent program instance. For more information see Section B.7.1, "Setup Configurator Data Migration" on page B-21.

7. Run the Migrate Configurator Data concurrent program from the target database instance. For more information, including possible issues output in the log file, see Section B.7.2, "Migrate Configurator Data" on page B-22.

8. When no issues or errors are found, run the Synchronize All Models concurrent program on the target database if the source database instance contained imported BOM data. The target instance must be synchronized after a

successful migration. For more information on BOM Model synchronization, see Section 7.2.1 on page 7-2.

Migration does not:

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

  Because there is typically a large amount of data and migration time associated with saved configurations, migrating saved configurations is not recommended.

7

Synchronizing Data

This chapter explains how to restore the identity and linkage of mismatched data by:

- Synchronizing BOM Data
- Synchronizing Publication Data

7.1 Overview

The kinds of data and circumstances requiring synchronization are:

- BOM Models
  - The import server changed to a different database instance
  - The production database instance is not the import server
  - 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 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 Chapter 6, "Migrating Data".

For information about synchronizing BOM data, see Section 7.2, "Synchronizing BOM Data".

For information about synchronizing publication records on cloned database instances see Section 7.3.1, "Synchronizing Publication Data after a Database Instance is Cloned" on page 7-6.

7.2 Synchronizing BOM Data

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

Data synchronization is not the same as data refresh (see Section 5.2.10, "Refreshing Imported Data" on page 5-15).

The concurrent programs for synchronizing BOM Model data are described in Section B.5, "Model Synchronization Concurrent Programs" on page B-16.

7.2.1 The BOM Synchronization Process

The process for synchronizing BOM data is as follows:

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

   For more information, see Section 7.2.2, "Checking BOM and Model Similarity" on page 7-3.

2. Synchronize the BOM in the configuration model with the source BOM by running the Synchronize All Models concurrent program. For more information, see Section 7.2.4, "Result of Synchronizing BOM Models" on page 7-5.

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

   - Reimport or refresh the BOM Model in the CZ schema (see Chapter 5, "Populating the CZ Schema")
   - Publish the configuration model (see Chapter 17, "Publishing Configuration Models")

   Running the publication concurrent programs includes BOM synchronization. For details, Section 17.3, "Publishing a Configuration Model" on page 17-10.

7.2.2 Checking BOM and Model Similarity

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

- Check Model / Bill Similarity
- Check All Models / Bills Similarity

For details about these concurrent programs, see Section B.5.1, "Check Model / Bill Similarity" on page B-16 and Section B.5.2, "Check All Models / Bills Similarity" on page B-17.

Running the Check Model / Bill Similarity and Check All Models / Bills Similarity concurrent programs creates a Check Model / Bill Similarity describing the fields that don't match and must be corrected before synchronization can occur. For more details, see Section 7.2.3, "Criteria for BOM Similarity" on page 7-3. For more information about the report see Section B.5.4, "Model / Bill Similarity Check Report" on page B-19.

7.2.3 Criteria for BOM Similarity

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

- 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 BOMs. For example, each imported parent node has the same imported children Items as in the BOM structure. The order of the children may be different.
- Certain Item characteristics are the same. For example, 'Required when parent is selected' Property, minimum / maximum default quantities.
- 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 (INVENTORY_ITEM_ID), 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 the same for the child node and the parent.

– If there is more than one child node with the given identity (INVENTORY_ITEM_ID), 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.

Table 7–1 lists the configuration model's data fields that must be synchronized with the import source BOM or publication target.

Table 7–1 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
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

Table 7–1 (Cont.) Fields That Must Be Synchronized

Table	Field	Import	Publication
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, 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 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. SeeSection 5.2.7.4, "Modifying EXPLOSION_TYPE" on page 5-10 for more information about the EXPLOSION_TYPE setting. BOM synchronization does not check the mandatory fields.

7.2.4 Result of Synchronizing BOM Models

After determining that the source BOM and the BOM-based configuration model are sufficiently similar based on the report generated by the Check Model/Bill Similarity and Check All Models/Bills Similarity concurrent programs, the BOM Models can be synchronized either by running the Synchronize All Models or the publication concurrent programs. See Section B.5.3, "Synchronize All Models" on page B-18.

Attempting to synchronize mismatched BOMs 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. During data import, the CZ schema is populated with the ORIG_SYS_REF identification of the source BOM. However, the same BOM in Bills of Material of two different database instances may have different ORIG_SYS_REF identification.

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

Since 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 and Check All Models/Bills Similarity 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 synchronization.

7.3 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

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 Section B.8, "Publication Synchronization Concurrent Programs" on page B-22.

See Chapter 17, "Publishing Configuration Models" for details about creating publications, and about the relationship between the publication data on the source and target database instances.

7.3.1 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 become overlapping

Publication data inconsistencies need to be resolved by updating data on both the cloned and the publication source or 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 synchronizes the publication data in the new cloned target database with the publication data on the source database.
- Synchronize Cloned Source Data synchronizes the publication data in the new cloned source database with the publication data on the target database.

See Section 7.3.2.4, "Example of Synchronizing Publication Data on a Cloned Target" on page 7-9 for details about the circumstances and results of synchronizing a cloned publication target. See Section 7.3.2.5, "Example of Synchronizing Publication Data on a Cloned Source" on page 7-11 for details about the circumstances and results of synchronizing a cloned publication source.

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

7.3.2 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.

7.3.2.1 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.

7.3.2.2 CZ_MODEL_PUBLICATIONS Table

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

- PUBLICATION_ID
- REMOTE_PUBLICATION_ID
- SERVER_ID

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 Oracle Configurator Developer Model Publishing window.

REMOTE_PUBLICATION_ID

REMOTE_PUBLICATION_ID on the source database instance points to the PUBLICATION_ID on the target database instance. Whereas the REMOTE_ PUBLICATION_ID on the target database instance points to the PUBLICATION_ID on the source database instance. See Figure 7–1, "Original Publication" on page 7-9.

SERVER_ID

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

7.3.2.3 Example Publication Data Before Cloning

The following explanations of example publication data presume a publication source database, A, with PUBLICATION_ID 1000 and a publication target database, B, with PUBLICATION_ID 2000. Figure 7–1 shows the original publication records on Source A and Target B.

In the publication record on Source A:

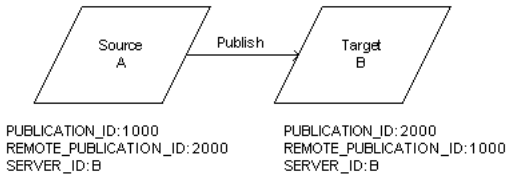
- REMOTE_PUBLICATION_ID is 2000 because it points to the PUBLICATION_ ID on the publication target
- SERVER_ID of the publication record is B because it points to the LOCAL SERVER_ID on the publication target

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

Figure 7–1 Original Publication



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

7.3.2.4 Example of Synchronizing Publication Data on a Cloned Target

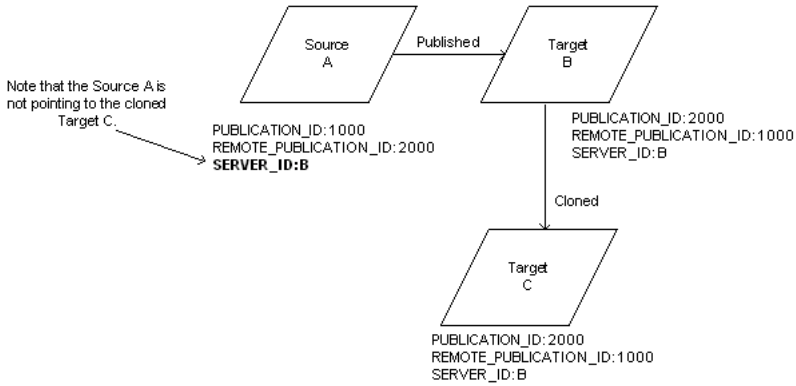
Synchronizing publication data on a cloned target resolves the following issues caused by cloning the publication target:

- The CZ_SERVERS table on the source does not include a listing for the cloned target
- A database link needs to 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.

Figure 7–1 shows the original publication records on Source A and Target B.

Target B is then cloned to create Target C. Figure 7–2 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).

Figure 7–2 Publication After Cloning



Source A is then synchronized with Target C. Figure 7–3 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.

Figure 7–3 Publication After Synchronization

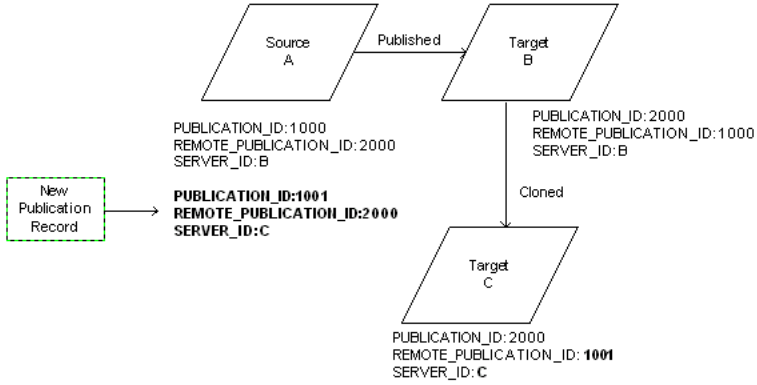


Table 7–2 summarizes the publication information from the original publication to the cloning, to the synchronization.

Table 7–2 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
REMOTE_ PUBLICATION_ID	2000	1000	1000
SERVER_ID	B	B	B
After Synchronizing Source A and Target C:			
PUBLICATION_ID	1000	2000	2000
REMOTE_ PUBLICATION_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 concurrent program, see Section B.8.1 on page B-23.

7.3.2.5 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 needs to 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. It is advisable to decommission the original source when synchronizing the cloned source.

Figure 7–4 illustrates a Model that is originally published from Source A to Target C.

Figure 7–4 Publication Before Cloning the Source Database

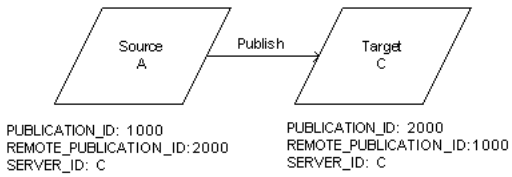


Table 7–3 illustrates some of the entries for database instances A and C in the CZ_SERVERS table of Source A before cloning.

Table 7–3 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

Table 7–4 illustrates some of the entries for database instances A and C in the CZ_SERVERS table of Target C before cloning.

Table 7–4 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

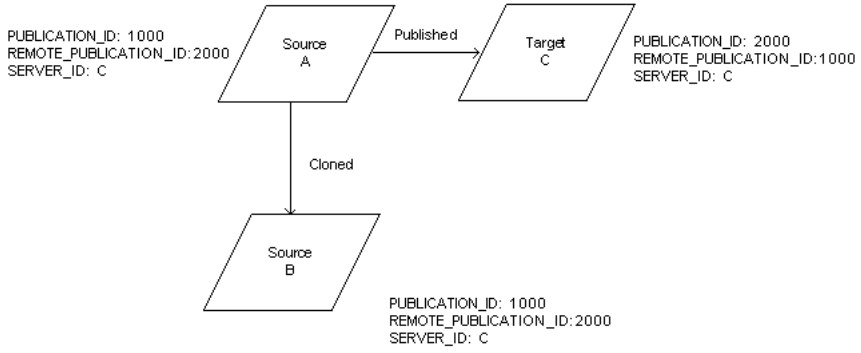
The SOURCE_SERVER_FLAG on Target C is set to 1, meaning Target C recognizes Source A as its publication source.

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 needs to 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

Figure 7–5 illustrates Source B as a clone of Source A.

Figure 7–5 Source Server B is Cloned from Source Server A



After cloning, the CZ_SERVERS table of the clone is an exact copy of the original Source A (see Table 7–3). Source B must be synchronized because its CZ_SERVERS table does not a LOCAL entry for Source B.

In order 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 concurrent program on Source B. See Section B.8.2, "Synchronize Cloned Source Data" for more information.

Running the Synchronize Cloned Source Data concurrent program updates the LOCAL entry in the CZ_SERVERS table on Source B with correct information. Table 7–5 shows the entries in the CZ_SERVERS table on B after running the Synchronize Cloned Source Data concurrent program.

Table 7–5 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. Table 7–6 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.

Table 7–6 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 *not* to decommission Source A, and there are configuration models that were published from A to C, then running the Synchronize Cloned Source Data concurrent program on Source B removes any cloned publications to

prevent conflict between the two publications sources and allow Source A to continue as the source for those publications.

8

CZ Schema Maintenance

Data that is maintained in more than one place is subject to becoming out of alignment. This chapter presents ways to help you keep multiple data sources in alignment.

8.1 Overview

Data that is maintained in more than one place is subject to becoming out of synch. Inventory and Bills of Material data must be maintained in the production instance. You can maintain the Oracle Configurator Schema with the data in the production instance by:

- Refreshing or Updating the Production Schema
- Eliminating any unused data by Purging Configurator Tables
- Restoring sequences from a dump file by Redoing Sequences
- Synchronizing BOM Data

8.2 Refreshing or Updating the Production Schema

When an runtime Oracle Configurator is deployed, the data is stored in the Oracle Configurator schema directly through networked use. During deployment, further imports are performed to refresh the Oracle Configurator schema as Oracle Applications or legacy data changes. The procedures that perform the import prevent customer-specific groups of fields in the Oracle Configurator 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 Oracle Configurator schema, see Section 5.2.10, "Refreshing Imported Data" on page 5-15.

8.3 Purging Configurator Tables

Performance is affected when databases get large. The Purge Configurator Tables concurrent program physically deletes all logically-deleted records in the tables and subschema.

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 EXPRESSION.
- 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.

When you run this concurrent program you can optionally delete information from the CZ_DB_LOGS and CZ_XFR_RUN_RESULTS tables based on the date or run identifier.

See Section B.1.3, "Purge Configurator Tables" on page B-4 for details on running this concurrent program.

8.4 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 subschema. The procedure checks the current high primary key value in the database and sets a new start value that is higher 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 Section 4.4.3.17, "OracleSequenceIncr" on page 4-15 for more information.

Part III

Integration

Part III presents integration information for setting up Oracle Configurator with other Oracle Applications or a custom application as described in Section 1.3, "Integration Tasks" on page 1-3. Part III contains the following chapters:

9

Session Initialization

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

> **Note:** If your host application is part of Oracle Applications, then the initialization message is already defined. You only need to implement an initialization message for custom host applications.

9.1 How it Works

In order to prepare a Model so that it can be configured in the Oracle Configurator window, you must first either:

- Import BOM and item data into the Oracle Configurator schema so that it is available to Oracle Configurator Developer. Importing is described in Chapter 5, "Populating the CZ Schema" on page 5-1.
- Use Oracle Configurator Developer to develop a Model and its associated Configuration Rules and a User Interface.

In order to integrate the Oracle Configurator window into your host application, the application must:

- Use frames, one of which contains the Oracle Configurator window.
- Provide a means of posting the initialization message.
- Provide a return URL servlet to process results of your user's selections in the Oracle Configurator window.

In a typical host application (such as a web store), a button, tab, or similar control is coded so that it causes the Oracle Configurator window to appear, and to contain

the appropriate user interface. For the purposes of this explanation, think of this control as "the Configure button". You intend for your user to click it at a point where configuration of a product is required.

Oracle Configurator provides you with:

- A **servlet**, the Oracle Configurator Servlet, that handles interaction between the host application, the Oracle Configurator window, and the configuration model that you define in Oracle Configurator Developer (which contains model structure, configuration rules, and user interface definitions).
- The host application invokes the URL of the OC Servlet with a query string that contains an **initialization message** for the session. It does this by setting the location property of the frame where the configuration window is to be displayed. You customize this initialization message to specify a number of important parameters that govern the behavior of the Oracle Configurator window. These parameters are described in this chapter.
- A set of **HTML Template files** that interact with the OC Servlet. You can customize these files to suit them to the needs of your host application, or leave them as they are. See Chapter 16, "Customizing the HTML Template" for details.

See Section 2.2, "Three-Tier Architecture" on page 2-5 in Chapter 2, "Configurator Architecture" for an explanation of the interaction between all these elements.

9.2 What You Do

Integrating the Oracle Configurator window with your host application consists primarily of causing your host application (e.g., through the coding of the Configure button) to post the XML initialization message to the OC Servlet.

You must first install the OC Servlet, as described in the *Oracle Configurator Installation Guide*. There is additional information on the files that are installed, in Chapter 12, "Custom Integration" of this document.

The basic situations that the host application must handle when integrating the Oracle Configurator window are shown in the following table:

You need to handle this...	As described in...
Initialization of the Oracle Configurator window, to prepare it for your user's configuration session.	Section 9.3, "Definition of Session Initialization" on page 9-3

You need to handle this...	As described in...
Termination of the Oracle Configurator window, to return control and results to the host application when your user closes the window.	Section 10.3, "Definition of Session Termination" on page 10-1

9.2.1 Responsibilities of the Host Application

The responsibilities of the host application while the Oracle Configurator window is in use are:

- Disable visible functions in the surrounding host application that would confuse the user while interacting with the Oracle Configurator window.
- Handle the output from the **return URL**, and close the configurator window by resetting its frame's location property.

9.3 Definition of Session Initialization

Session initialization takes place when your host application invokes the OC Servlet which opens the Oracle Configurator window.

When you set the parameters of the initialization message in your host application, your parameters handle the types of responsibilities listed in Section 9.5, "Initialization Parameter Types" on page 9-8.

When your host application invokes the Oracle Configurator window, the initialization message is sent to the OC Servlet, using the HTTP POST method. (POST is used in preference to GET to accommodate the length of the message.)

The initialization message is written in XML, and has `<initialize>` as its document element. You must specify the parameters for `<initialize>` in order to determine the state in which the Oracle Configurator window opens.

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*.

9.4 Setting Parameters

You specify <initialize> and its parameters as the value of an XML message that is passed to the OC Servlet. The OC Servlet is invoked through its URL, which is determined when you install it (as described in the *Oracle Configurator Installation Guide*). By default, the URL is:

hostname:*port*/*virtual_path_of_servlet_class*/oracle.apps.cz.servlet.UiServlet

For example:

http://www.mysite.com:8802/configurator/oracle.apps.cz.servlet.UiServlet

For use with Oracle Configurator, the *virtual_path_of_servlet_class* always set to configurator.

9.4.1 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:

<param name="*parameter_name*">*parameter_value*</param>

Example 9–1 shows the basic syntax for specifying the OC 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.

Example 9–1 Syntax of initialization message in HTML context

```
...
<form action="servlet_URL" method="post" >
<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>'>
<input type="submit" value="Configure">
</form>
...
```

When your user clicks the Configure button produced by the second INPUT element in Example 9–1, the initialization message is posted to the OC Servlet.

See Example 9–2 for some typical values for the parameters, and Example 9–3 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:

```
"<initialize>
    <param name='parameter_name'>parameter_value</param>
</initialize>"
```

9.4.1.1 Omitting Parameters and Values

If you omit a parameter entirely from the initialization message, then the parameter is ignored by the OC Servlet.

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 Section 9.6, "Initialization Parameter Descriptions" on page 9-16.

> **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 OC Servlet 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.

9.4.2 Typical Parameter Values

Example 9–2 shows an example of a basic set of initialization parameters that cover all of the types of responsibility shown in Table 9–2 on page 9-8.

See Section 9.6, "Initialization Parameter Descriptions" on page 9-16 for the complete list of valid parameters to the initialization message.

See Example 9–1 for the syntax of the initialization message, and Example 9–3 for a test page that puts the values in context.

Example 9–2 Basic XML initialization parameters

```
<initialize>
  <param name="two_task">vis11</param>
  <param name="gwyuid">applsyspub/pub</param>
  <param name="fndnam">apps</param>
  <param name="user">mfg</param>
  <param name="pwd">welcome</param>
```

```
  <param name="ui_type">DHTML</param>
  <param name="ui_def_id">3120</param>
  <param name="return_
url">http://www.mysite.com:10130/configurator/Checkout</param>
</initialize>
```

Table 9–1 explains the parameters used in Example 9–2.

Table 9–1 Explanation of initialization parameters in Example 9–2

Parameter type	Name	Description
Login	two_task	The name of the database instance to connect to. This value can be read from the environment variable TWO_ TASK.
Login	gwyuid	The gateway user ID required for authentication with the Oracle Applications protocol.
Login	fndnam	Used by Oracle Configurator to authenticate standard Oracle Applications users through the FND login algorithms.
Login	user	The user ID of the login user.
Login	pwd	The password of the login user.
Login	ui_type	The type of web pages to be returned by the OC Servlet.
Configuration Identification	ui_def_id	The UI Definition ID of the User Interface that you created in Oracle Configurator Developer. There are several ways to specify a Configuration choice. See Section 9.5.3, "Model Identification Parameters" on page 9-10.
Return	return_url	The URL of the Java servlet that processes the configurator's termination message.

9.4.3 Minimal Test of Initialization

Example 9–3 shows the HTML for a minimal web page that invokes the Oracle Configurator window. You can use this test page as a stand-in for your host application.

This example includes the invocation of the OC Servlet as shown in Example 9–1 and the initialization message parameters as shown in Example 9–2.

Example 9–3 Minimal HTML for invoking the Oracle Configurator window

```
<html>
```

```
<head>
<title>Minimal Configurator Test</title>
</head>
<body>
<form
action="http://www.mysite.com:10130/configurator/oracle.apps.cz.servlet.UiServle
t" method="post">
<input type="hidden" name="XMLmsg" value=
'<initialize>
<param name="two_task">vis</param>
<param name="gwyuid">applsyspub/pub</param>
<param name="fndnam">apps</param>
<param name="user">mfg</param>
<param name="pwd">welcome</param>
<param name="ui_type">DHTML</param>
<param name="ui_def_id">3120</param>
<param name="return_
url">http://www.mysite.com:10130/configurator/Checkout</param>
</initialize>'>
<p>Click button to configure model...
<input type="submit" value="Configure">
</form>
</body>
</html>
```

9.4.4 Parameter Validation

When your host application invokes the OC Servlet, the UI Server validates the parameters of the initialization message.

- There must be a way of connecting to the database, such as the parameters `two_task` or `alt_database_name`.
- There must be a way to choose a Model to be configured, so the initialization message must include one of the combinations described in Section 9.5.3, "Model Identification Parameters" on page 9-10.

If there is a problem processing the initialization message, then the UI Server returns a termination object to the OC Servlet, which returns it to the host application or displays the results to your user, through the URL specified in the `return_url` parameter.

9.4.5 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. The location and naming of these log files is controlled with the OC Servlet property `cz.uiservlet.logfilename`. See the *Oracle Configurator Installation Guide* for more information.

9.5 Initialization Parameter Types

This section describes the use of the types of initialization parameters listed in Table 9–2 on page 9-8. All of the initialization parameters are described alphabetically in Section 9.6, "Initialization Parameter Descriptions" on page 9-16.

Table 9–2 Types of Initialization Parameters

Type	Required?	Description	See
Login	Yes	Information required for access to the proper data, such as database, user, and password.	Section 9.5.1, "Connection Parameters" on page 9-9
Publication	Yes	Information required to select the correct Model publication.	Section 9.5.2, "Model Publication Identification Parameters" on page 9-9
Configuration	Yes	Identification of the Model to be configured, or of the existing configuration to be modified.	Section 9.5.3, "Model Identification Parameters" on page 9-10
Return	No, but recommended	Identification of the return URL that handles the results from the Oracle Configurator window, such as configuration outputs.	Section 9.5.4, "Return URL Parameter" on page 9-12
Pricing and ATP	No	Identification of the procedures and interfaces to be used for obtaining prices and ATP dates.	Section 9.5.5, "Pricing Parameters" on page 9-13 Section 9.5.6, "ATP Parameters" on page 9-14

Table 9–2 (Cont.) Types of Initialization Parameters

Type	Required?	Description	See
Other	No	Miscellaneous desired information.	Section 9.5.7, "Arbitrary Parameters" on page 9-15

9.5.1 Connection Parameters

In order to connect the Oracle Configurator window to the database, you must specify one of the following parameters or combinations of parameters in your initialization message.

- database_id
- database_id and icx_session_ticket
- alt_database_name, user, and pwd
- two_task, user, pwd, gwyuid, and fndnam

For descriptions of the individual parameters, see Section 9.6, "Initialization Parameter Descriptions" on page 9-16.

9.5.2 Model Publication Identification Parameters

To determine the Model publication to display, you must specify in your initialization message one or more of the parameters listed in Table 9–3 on page 9-9.

These initialization parameters correspond to the applicability parameters that you specify when creating the publication in the Model Publishing window in Oracle Configurator Developer. See the *Oracle Configurator Developer User's Guide* for more information.

Table 9–3 Applicability Parameters for Publishing

Initialization Parameter	OCD Publishing Parameter
calling_application_id	Applications
config_effective_usage	Usages
config_model_lookup_date	Valid From / Valid To
publication_mode	Mode

9.5.3 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 Table 9–4:

Table 9–4 Configuration Identification Parameters

Method for Configuration Identification	Initialization Parameters
Section 9.5.3.1, "Identifying the User Interface Definition"	ui_def_id
Section 9.5.3.2, "Identifying the Configuration"	config_header_id config_rev_nbr
Section 9.5.3.3, "Identifying the Model"	Imported BOM Models: organization_id inventory_item_id
	Configurator Developer Models: product_id
Section 9.5.3.4, "Support of Multiple Instantiation"	sbm_flag

For detailed descriptions of the individual parameters, see Section 9.6, "Initialization Parameter Descriptions" on page 9-16.

9.5.3.1 Identifying the User Interface Definition

Parameter to specify:

- ui_def_id

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).

9.5.3.2 Identifying the Configuration

Parameters to specify:

- config_header_id
- config_rev_nbr

Using this combination of parameters restores an existing 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 Oracle Configurator schema, such as the runtime Oracle Configurator. The Configuration Revision Number distinguishes among particular saved configurations sharing the same header information.

9.5.3.3 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:

- organization_id
- inventory_item_id

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 instead of organization_id
- model_id instead of inventory_item_id

Models Created in Configurator Developer

Parameters to specify:

- product_id
- publication_mode (for custom applications only)
- config_effective_usage (for custom applications only)

If you created a Model in Oracle Configurator Developer, and entered a Product ID when you published the Model, then you should specify only the product_id in your initialization message to identify the model to configure.

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 and config_effective_usage. If the host application is part of Oracle Applications (such as Order Management), then the Usage and Mode are obtained from the profile options CZ:Publication Usage and CZ:Publication Lookup Mode; consequently, you do not have to pass Usage and Mode in the initialization message.

9.5.3.4 Support of Multiple Instantiation

- sbm_flag

During runtime, the presence of this flag is checked to see if the calling 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 informing the end user that the calling application does not support multiple instantiation is returned.

9.5.4 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 after the user has ended the configuration session.

- return_url

Here is an example of the use of this parameter, from Example 9–3 on page 9-6:

```
<param name="return_
url">http://www.mysite.com:10130/configurator/Checkout</param>
```

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 instance, 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.

The return URL servlet is installed in your web server's Servlet directory, which is described in Section 12.1.2, "Files for the Servlet Directory" on page 12-2.

See Section 10.8, "The Return URL" on page 10-10 for details on the implementation of the return servlet.

9.5.5 Pricing Parameters

See Chapter 13, "Pricing and ATP in Oracle Configurator" for details on the use of these parameters. See Section D.1, "Pricing and ATP Callback Procedures" on page D-2 for examples.

Choose these pricing parameters, depending on which pricing methods are required for your host application, as indicated in the following table:

Pricing method	Use these parameters...
Advanced Pricing (QP), or your own callback pricing procedures that call it	pricing_package_name and the associated parameters listed under Section 9.5.5.1, "Oracle Applications Release 11i Pricing Parameters" on page 9-13

For descriptions of the individual parameters, see Section 9.6, "Initialization Parameter Descriptions" on page 9-16.

9.5.5.1 Oracle Applications Release 11*i* Pricing Parameters

Since these parameters are designed to be used with an interface using callback procedures, they are also referred to as **callback pricing parameters**.

To use callback pricing, provide these parameters:

- pricing_package_name
- configurator_session_key
- either price_mult_items_proc, price_mult_items_mls_proc, or price_single_item_proc

These parameters are used when the Oracle Configurator window calls existing APIs to get pricing and ATP data for configured items.

These parameters are accessible with the Configuration Interface Object (CIO) method `Configuration.setInitParameters()`, described in the API reference portion of the *Oracle Configuration Interface Object (CIO) Developer's Guide*.

9.5.6 ATP Parameters

See Chapter 13, "Pricing and ATP in Oracle Configurator" for details on the use of these parameters. See Section D.1, "Pricing and ATP Callback Procedures" on page D-2 for examples.

Choose from the ATP (Available To Promise) parameters, depending on which ATP methods are required for your host application, as indicated in the following table:

ATP method	Use these parameters...
callback interface	atp_package_name and the associated parameters listed under Section 9.5.6.1, "Oracle Applications Release 11i ATP Parameters" on page 9-14

For descriptions of the individual parameters, see Section 9.6, "Initialization Parameter Descriptions" on page 9-16.

9.5.6.1 Oracle Applications Release 11*i* ATP Parameters

Since these parameters are designed to be used with an interface using callback procedures, they are also referred to as **callback ATP parameters**.

To use callback ATP, provide these parameters:

- atp_package_name
- configurator_session_key
- get_atp_dates_proc
- requested_date (optional, defaults to SYSDATE)
- warehouse_id
- and one of the following:
  - customer_id and customer_site_id

- ship_to_org_id

9.5.7 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 would specify the arbitrary parameter as a name-value pair, using the syntax described in Section 9.4.1, "Parameter Syntax" on page 9-4:

```
<param name="parameter_name">parameter_value</param>
```

For 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 Functional Companions. (See the *Oracle Configuration Interface Object (CIO) 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.

9.5.8 Parameter Compatibility

The initialization message parameters for this release are backwardly compatible. A host application can continue to use the initialization message parameters provided for the previous release with the same results, unless a parameter has been replaced or withdrawn, thus making it obsolete.

Obsolete parameters are listed in Section 9.5.9, "Obsolete Parameters" on page 9-15.

9.5.9 Obsolete Parameters

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 your application.

- agreement_id
- agreement_type_code
- gsa

- invoice_to_site_use_id
- order_type_id
- po_number
- price_list_id
- responsibility_id

9.6 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 Section 9.4, "Setting Parameters" on page 9-4. The parameters are summarized in Table 9–5.

Table 9–5 Initialization Parameters for Oracle Configurator

Name
alt_database_name
application_id
apps_connection_info
atp_package_name
calling_application_id
client_header
client_line
client_line_detail
config_creation_date
config_effective_date
config_effective_usage
config_header_id
config_model_lookup_date
config_rev_nbr
configurator_session_key
context_org_id
customer_id

Table 9–5 (Cont.) Initialization Parameters for Oracle Configurator

Name
customer_site_id
database_id
fndnam
get_atp_dates_proc
gwyuid
icx_session_ticket
inventory_item_id
model_id
model_quantity
organization_id
price_mult_items_mls_proc
price_mult_items_proc
price_single_item_proc
pricing_package_name
product_id
publication_mode
pwd
read_only
requested_date
return_url
save_config_behavior
sbm_flag
ship_to_org_id
template_url
terminate_id
terminate_msg_behavior
two_task

Table 9–5 (Cont.) Initialization Parameters for Oracle Configurator

Name
ui_def_id
ui_type
user
user_id
warehouse_id

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. Recommended for use during development of your application, as an alternative to connecting as an Oracle Applications user. Not recommended for production deployment. Must specify thin drivers, for example: `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 (e.g., Release 11*i*), 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
- database_id and icx_session_ticket
- user, pwd, gwyuid, fndnam, and two_task
- alt_database_name, user, and pwd

atp_package_name

The name of the PL/SQL interface package that the OC Servlet 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.

calling_application_id

The ID from FND_APPLICATION.APPLICATION_ID that is the ID of the host application. See the *Oracle Configurator Release Notes* for a list of Oracle Applications that host Oracle Configurator.

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 Applications System Administrator's Guide*.)

If the hosting application is part of Oracle Applications (for example, Order Management, *i*Store, or Telesales), it is important to note that the hosting application will display 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 know 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.

Required.

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 *Oracle Configurator Methodologies*. See also client_line and client_line_detail.

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 *Oracle Configurator Methodologies*. See also client_header and client_line_detail.

client_line_detail

A string or number used to provide additional information if client_line does not provide enough. Used in conjunction with the methodology for input configuration attributes, which is described in *Oracle Configurator Methodologies*. See also client_header and client_line.

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.

The value must be in the format `MM-DD-YYYY-HH-MM-SS`. The values for the tokens in this format are shown in Table 9–6:

Table 9–6 Date and Time Format for config_creation_date Parameter

Token	Meaning
MM	The number of the month
DD	The number of the day of the month
YYYY	The year
HH	The 24-hour representation of the hour
MM	The number of minutes
SS	The number of seconds

Example `<param name="config_creation_date">03-25-2001-19-30-02</param>`

Defaults For a new configuration: the value of SYSDATE. For a restored configuration: the saved value of config_creation_date. If the parameter value does not include the `HH-MM-SS` portion, then the default time is assumed to be midnight (`00-00-00`).

config_effective_date

The date used to filter effective nodes and rules.

This parameter has the same structure as config_creation_date.

Defaults For a new configuration: the value of config_creation_date. For a restored configuration: the saved value of config_effective_date.

Not required.

config_effective_usage

The publishing Usage name. The value is not case-sensitive.

Determines the publishing Usage name for the configuration Model. If this parameter is missing, Oracle Configurator checks the Oracle Applications profile option CZ:Publication Usage. If this profile option is not defined, the default value of this profile option, which is `Any Usage`, is used.

Default The default value is `Any Usage`, unless CZ:Publication Usage has been set.

Not required.

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 Oracle Configurator 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.

Defaults For a new configuration: the value of config_creation_date. For a restored configuration: the saved value of config_effective_date.

Not required.

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 Oracle Configurator schema.

configurator_session_key

An application-dependent string that identifies a configuration session, and allows linking a pricing or ATP request from the Oracle Configurator window 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.

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.

customer_site_id

When getting ATP dates, the ID of the customer site to which the configured product is to be shipped.

database_id

The name of a DBC file that contains database connectivity information. This file can be found in a standard Oracle Applications installation by calling the PL/SQL function `fnd_web_config.database_id`.

fndnam

Used to establish an Oracle Applications context. Its value can be read from the Forms environment variable FNDNAM. Oracle Configurator authenticates

standard Oracle Applications users by using the FND login algorithms. Used in conjunction with gwyuid.

get_atp_dates_proc

The name of the "get ATP dates" procedure to be called from the package specified by atp_package_name. This parameter is conditionally required; it must be provided if the ATP callback interface is to be used.

gwyuid

Used to establish an Oracle Applications context. Its value can be read from the Forms environment variable GWYUID. Used in conjunction with fndnam.

icx_session_ticket

An ICX session ticket encodes an Oracle Applications session.

This is the recommended way for Oracle Applications to call the Oracle Configurator window.

You should 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 on page 18-45 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.

inventory_item_id

This parameter is a synonym that replaces model_id.

This parameter is the imported Inventory Item ID for the top-level imported BOM Model. It is used together with organization_id 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.

Conditionally required. No default.

model_id

This parameter is for backward compatibility only. Instead of this parameter you should use its synonym, inventory_item_id.

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

The value of this parameter is a number that indicates how many 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 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.
- 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 Table 9–7, " Effects of Contributions to Model Quantity" on page 9-25. Not all of the possible scenarios are illustrated.

In Table 9–7, 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.

Table 9–7 Effects of Contributions to Model Quantity

	Contribution	Model Quantity	Final Quantity
New Configuration:			
Case 1	C	NM>=C	NM
Case 2	C	NM<C	C, with Validation Failure[1]
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

[1] These Validation Failure messages are deleted once their text is viewed.

organization_id

This parameter is a synonym that replaces context_org_id.

This parameter is the imported Organization ID for the top-level imported BOM Model. It is used together with inventory_item_id 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 Applications Order Management, this is the organization identifier for the BOM exploder. The value should be the same as the profile option OM: Item Validation Organization.

If you are using a multiple organization structure, your system administrator must change the OM: Item Validation Organization parameter to be visible and updatable at the responsibility level. This change allows Order Management to default code and revenue account information accurately. Note that the Organization ID is not the same as the Warehouse ID.

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 calling application that supports multiple currencies, not just USD (US dollars).

This parameter is conditionally required; one of this parameter, price_single_item_proc, or price_mult_items_proc must be provided if pricing callbacks are to be used.

You should use this parameter in preference to price_mult_items_proc, since the procedure called through this parameter displays prices in the right currency and the right format.

price_mult_items_proc

The name of the "price multiple items" procedure to be called from the package specified by pricing_package_name.

This parameter is conditionally required; one of this parameter, price_single_item_proc, or price_mult_items_mls_proc must be provided if pricing callbacks are to be used.

You should use price_mult_items_mls_proc in preference to this parameter, since the procedure called through this parameter displays prices only in USD (US dollars).

This parameter takes precedence over price_single_item_proc.

price_single_item_proc

The name of the "price single item" procedure to be called from the package specified by pricing_package_name.

This parameter is conditionally required; one of this parameter, price_mult_items_proc, or price_mult_items_mls_proc must be provided if pricing callbacks are to be used.

This procedure is not called if price_mult_items_proc is provided.

Deprecated This parameter is now deprecated; use price_mult_items_proc if possible.

pricing_package_name

The name of the PL/SQL interface package that the OC Servlet 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 or price_single_item_proc.

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 and inventory_item_id instead of this parameter.

If this parameter is included in the initialization message, the OC Servlet uses the function CZ_CF_API.CONFIG_MODEL_FOR_PRODUCT 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 Oracle Configurator 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 and config_effective_usage. If the host application is part of Oracle Applications (such as Order Management), then the Usage and Mode are obtained from profile options CZ:Publication Usage and CZ:Publication Lookup Mode.

Defaults Conditionally required. No default.

Examples To make your application use a Configurator Developer Model with the name `Test Model`, insert the following parameter in your initialization message:

```
<param name="product_id">Test Model</param>
```

To make your application use an imported BOM Model with the organization_id 204 and the inventory_item_id 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. If this parameter is missing, Configurator checks the Oracle Applications profile option CZ:Publication Lookup Mode. If this option is not defined, Production mode (P) is used to look up the publication.

The values allowed for this parameter are shown in the following table:

Value	Meaning
P	Production
T	Test

Default The default value is P.

Not required.

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 two_task parameter. Use the database password if you identified the database with the alt_database_name parameter. Used in conjunction with user.

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 Done button is disabled. The UI Server displays a message at the beginning of the configuration session, indicating that the session is read-only.

Default `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.

return_url

The fully qualified URL of a Java servlet installed on your Web server that implements the behavior desired after a configuration session is terminated. Used only when the runtime Oracle Configurator is Dynamic HTML in a browser. See Section 9.5.4, "Return URL Parameter" on page 9-12 for details.

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 `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.

Value	Meaning
`True`	The hosting application has installed the appropriate software patch that supports multiple instantiation.

Value	Meaning
False	The software patch supporting multiple instantiation has not been installed, and multiple instantiation is not supported by the hosting application.

A message is returned when an end user attempts to instantiate a component at runtime, and the hosting application does not support instantiation. If the sbm_flag is not passed at all, hosting application support of multiple instantiation is considered False.

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

The URL of the template file that the Oracle Configurator window uses when displaying its initial state. If there need to be multiple templates for multiple languages or browsers, it is the responsibility of the host application to choose the correct template. The web page pointed to by the template URL must contain the content frame and the proxy frame. You may need to account for language-specific installation directory names, such as `OA_HTML/US`, when specifying this parameter. Used only when the runtime Oracle Configurator is Dynamic HTML in a browser. See Chapter 16, "Customizing the HTML Template" for information on template files, and Section 16.1.1.5 on page 16-8 for background on using this parameter.

Example To use the template file `OA_HTML/US/myFrame.htm`, add the following parameter to the initialization message:

```
<param name="template_url">http://host:port/OA_HTML/US/myFrame.htm</param>
```

Defaults For a user interface generated with the Oracle Web Look (also called browser look and feel (BLAF)): `czBlafTemplate.htm`. For a user interface generated with the Oracle Forms Look: `czFormTemplate.htm`.

terminate_id

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 Chapter 13).
`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 will not be readable from the database. If Oracle Configurator gets 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.

two_task

The name of the database instance to connect to. This value can be read from the environment variable TWO_TASK.

ui_def_id

The identifier for the User Interface created in Configurator Developer. The value for the `ui_def_id` parameter is obtained from CZ_UI_DEFS.UI_DEF_ID in the Oracle Configurator schema. The value can also be obtained by calling the PL/SQL function `cz_cf_api.ui_for_item`.

ui_type

Type of client. The only supported values are `Applet` and `DHTML`. Must be set to `DHTML` when using the Oracle Configurator window in a custom Web deployment.

user

The username to use when logging in. Use the Oracle Applications username if you identified the database with the two_task parameter. Use the database username if you identified the database with the alt_database_name parameter. Used in conjunction with pwd.

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 Oracle Configurator Servlet.

> **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.

10.1 How it Works

See Section 9.1, "How it Works" on page 9-1, in Chapter 9, "Session Initialization".

10.2 What You Do

See Section 9.2, "What You Do" on page 9-2, in Chapter 9.

10.3 Definition of Session Termination

Session termination takes place when the Oracle Configurator window is closed by one of the conditions listed in Table 10–1.

Table 10–1 Termination conditions

Condition	Example	Explanation
Submission	Your user clicks the Done button.	See Section 10.5, "Submission" on page 10-3
Cancellation	Your user clicks the Cancel button.	See Section 10.6, "Cancellation" on page 10-9

Table 10–1 (Cont.) Termination conditions

Condition	Example	Explanation
Error	A connection cannot be made to the database.	See Section 10.7, "Error" on page 10-9

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 in order to be able to extract the desired data from it in your return URL servlet. The structure of this message is described in Section 10.4, "XML Message Structure" on page 10-2.

10.4 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:

```
<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:

```
...
<!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*)>
...
```

Example 10–1 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.

Example 10–1 Example of 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>
```

10.5 Submission

Submission occurs after your user closes the Oracle Configurator window by clicking the Done button.

The meaning of the Done 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.

The Done button can be customized, as described in Section 16.1.3, "The Action Buttons" on page 16-14, in Chapter 16, "Customizing the HTML Template".

When the Done button is clicked, the OC Servlet determines whether a return URL has been specified. If so, the servlet it identifies is executed, 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 Section 10.8, "The Return URL" on page 10-10

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:

- Section 10.5.1, "Configuration Status"
- Section 10.5.2, "Configuration Outputs"
- Section 10.5.3, "Configuration Messages"

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. See the description of that parameter for details.

10.5.1 Configuration Status

The current configuration status is described by the subelements of `<terminate>` listed in this section.

10.5.1.1 Subelements for Configuration Status

config_header_id

The main identifier of an existing configuration. See the description for config_header_id on page 9-21. 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 on page 9-22. This value is displayed in the Oracle Configurator window with the default label "Configuration Revision".

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".

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 this 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 this 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 Done, or if the user changed the configuration and did not click the Summary button before clicking Done. In this case, the host application should reprice each configuration line, to ensure that the proper prices are applied to the configuration.

10.5.2 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.

Example 10–2 shows an example of configuration outputs in the termination message, with typographical emphasis and comments added.

Example 10–2 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>
```

10.5.2.1 Subelements for Configuration Outputs

selection_line_id

Contains the ID of the configuration line. It is the same as CZ_CONFIG_ITEMS.CONFIG_ITEM_ID in the Oracle Configurator schema.

component_code

Contains a value extracted from BOM_EXPLOSIONS.COMPONENT_CODE.

parent_line_id

Contains the value from CZ_CONFIG_ITEMS.CONFIG_ITEM_ID for the parent node. The is 0 (zero) for the root.

quantity

Contains the selected quantity for the option.

inventory_item_id

Contains the ID for the item, extracted from MTL_SYSTEM_ITEMS.INVENTORY_ITEM_ID.

organization_id

Contains the organization ID for the item, extracted from MTL_SYSTEM_ITEMS.ORGANIZATION_ID

uom

Contains the unit of measure.

atp_date

Contains the ATP date. This is calculated by using the ATP procedure specified in the initialization message. See Section 9.5.6, "ATP Parameters" on page 9-14, and Chapter 13, "Pricing and ATP in Oracle Configurator".

list_price

Contains the list price for the selected option. This is calculated by using the pricing procedure specified in the initialization message. See Section 9.5.5, "Pricing Parameters" on page 9-13, and Chapter 13, "Pricing and ATP in Oracle Configurator".

discounted_price

Contains the discounted price for the selected option. This is calculated by using the pricing procedure specified in the initialization message. See Section 9.5.5, "Pricing Parameters" on page 9-13, and Chapter 13, "Pricing and ATP in Oracle Configurator".

10.5.3 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.

See Section 10.7, "Error" on page 10-9 for details on how to handle validation failures.

Example 10–3 shows an example of a configuration message in the termination message, with typographical emphasis and comments added.

Example 10–3 Configuration messages in the termination message

```
<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>
```

10.5.3.1 Subelements for Configuration Messages

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_type

Contains the severity level of the message. Possible values include the following:

suggestion
warning
overridable error
error
autoselection
autoexclusion

not satisfied

message_text

Contains the text of the message.

10.6 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`:

```
<terminate>
  <exit>cancel</exit>
</terminate>
```

10.7 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 Section 9.5.1, "Connection Parameters" on page 9-9)
- Lack of any configuration parameters (see Section 9.5.3, "Model Identification Parameters" on page 9-10)
- Incorrect type for a parameter

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 Configuration Interface Object (CIO) Developer's Guide* for details.

Control is returned to the host application, and no configuration information is returned. Any validation failures are returned as messages in the `<config_messages>` element. The termination message contains the `<exit>` subelement, with a value of `error`:

```
<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>
```

10.8 The Return URL

The Java servlet specified by the return URL initialization parameter (return_url) determines how your host application uses the configuration information produced by your user's selections during a session in the Oracle Configurator window. The return URL servlet is executed upon termination of a configuration session, if you have specified the return URL in your initialization message for the Oracle Configurator window.

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.

10.8.1 Specifying the Return URL

You specify the identity of your return URL servlet in the XML initialization message, as the value of the parameter `return_url`:

```
<param name="return_
url">http://www.mysite.com:10130/configurator/Checkout</param>
```

The example parameter above comes from Example 9–3, "Minimal HTML for invoking the Oracle Configurator window" on page 9-6.

See also:

- Section 9.5.4, "Return URL Parameter" on page 9-12

- return_url on page 9-29
- Section 9.4.1, "Parameter Syntax" on page 9-4

10.8.2 Implementing the Return URL

See Example D–4 in Appendix D for an example of a return URL servlet. You can modify this servlet code for the needs of your host application.

In order to use some of the configuration information returned in the termination message (for instance, the outputs described in Section 10.5.2, "Configuration Outputs" on page 10-6), you can write a method that obtains the value of an element in the termination message by using the `getTagValue()` method of the Checkout servlet.

11

Batch Validation

This chapter describes using Oracle Configurator in programmatic mode.

Note: Batch validation operates only on options that are BOM items in Oracle Applications, so your host application must be part of Oracle Applications. You do not implement batch validation for custom host applications.

11.1 How it Works

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 user interface other than those provided with Oracle Configurator (that is, other than the Java applet and DHTML interfaces).

A host application calls batch validation through the CZ_CF_API.VALIDATE PL/SQL procedure (see Section 11.3 on page 11-4). This procedure passes the batch validation message to the URL of the OC Servlet (see Section 11.2 on page 11-2).

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 Table 11–1 on page 11-2.

An example of the batch validation message is provided in Example 11–1 on page 11-3.

Table 11–1 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.
`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. ▪ `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">`

Table 11–1 (Cont.) Elements of the Batch Validation Message

Element	Description
<initialize>	Described in Chapter 9, "Session Initialization". The parameters of the initialization message are described in Section 9.6, "Initialization Parameter Descriptions" on page 9-16. See the description of the database_id parameter on page 9-22 for connectivity information.
<config_inputs>	Composed of a list of <option> elements.
<option>	Described in Chapter 10, "Session Termination". When an <option> element is used in a <config_inputs> element, only the <component_code> and <quantity> elements of the <option> are used.

Example 11–1 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>
```

11.3 Calling the CZ_CF_API.VALIDATE Procedure

If the host application is written in PL/SQL, it should call the VALIDATE procedure. There are restrictions in the way that PL/SQL can request data from a URL that require 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 on page 18-61, in Chapter 18, "Programmatic Tools for Development".

Example 11–2 on page 11-4 shows fragments from a PL/SQL program that calls CZ_CF_API.VALIDATE.

Example 11–3 on page 11-5 shows a PL/SQL script that calls CZ_CF_API.VALIDATE.

Example 11–2 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 ,
              p_config_rev_nbr      IN NUMBER ,
              p_model_qty           IN NUMBER ,
              p_creation_date       IN DATE ,
              p_deleted_options_tbl IN   App_Order.request_tbl_type
                                    := App_Order.G_MISS_REQUEST_TBL,
              p_updated_options_tbl IN   App_Order.request_tbl_type
                                    := App_Order.G_MISS_REQUEST_TBL,
              x_out_XML_msg         OUT LONG ,
              x_return_status       OUT VARCHAR2 )
...
      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 );
```

Example 11–3 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 calling 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/configurator/oracle.apps.cz.servlet.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) -- 40, to fit alt_
database_name
                          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);  -- note extra length

--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',
                '&modelId',  -- 143 is the usual value
                null,
                null,
                null,
                null,
                'new_revision',
                null,
                '45',
                'ap505dbs_dom1151',
                '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,validation_
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;
/
```

12

Custom Integration

In order to customize Oracle Configurator in your host application, you may need to modify certain Oracle Configurator files. This chapter describes the location and purpose of those files.

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 the needs of your site or host application. If so, see Section 12.1, "Required Files and Locations" on page 12-1.

12.1 Required Files and Locations

There may be reasons for moving certain installed files to alternate locations.

- To suit the needs or working conventions of your site.
- To suit the needs of your host application.

In order to assist you in moving these files correctly, this section describes constraints and guidelines on their location.

The files are also enumerated, in order to assist you in troubleshooting simple problems arising from missing or misplaced files.

12.1.1 General Directory Structure

Table 12–1 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.

Table 12–1 General Structure of Directories for Oracle Configurator

Directory Role	Description
OC Installation	This directory is referenced in these instructions as *oc_install*. The directory in which you install OC, based on your choice of installation directory in the Oracle Configurator setup program. On Solaris, this defaults to $APPL_TOP.
Servlet	Contains the Java class or archive files that implement the OC Servlet.
HTML	Contains the HTML template files that define the runtime Oracle Configurator of your host application (see Chapter 16).
Media	Contains the image files used by the runtime Oracle Configurator of your host application.
Log	Contains log files written by the OC Servlet when the runtime Oracle Configurator is used.

Note that it is not strictly necessary for the Servlet directory to have a separate physical location, since the files it contains are referenced by environment variables that you set while installing the runtime Oracle Configurator servlet.

Directory and file names are case-sensitive on Solaris.

12.1.2 Files for the Servlet Directory

Table 12–2 shows the files that should be installed in your Servlet directory.

The Servlet directory contains files that must be referenced in the PATH and CLASSPATH environment variables. It is not strictly necessary for the Servlet directory to have a distinct physical location. For instance, its files can be located directly in the *oc_install* directory, rather than in a separate /servlets subdirectory.

Table 12–2 Files for the Servlet Directory

File	For Platform	Comment
Java Classes		
apps.zip	All	Includes all the Java classes required for the runtime Oracle Configurator, in addition to those for Oracle Applications.
xmlparserv2.zip	All	Includes the Java classes for the XML parser.

Table 12–2 (Cont.) Files for the Servlet Directory

File	For Platform	Comment
Shared Object		
`czlce.dll`	Windows NT	Must be in the PATH system environment variable on the host machine on which the servlet is installed. This should be set by the OC installation program.
`libczlce.so`	Solaris	Must be in the LD_LIBRARY_PATH environment variable parameter for your servlet.

12.1.3 Files for the HTML Directory

By default, the HTML directory is the directory pointed to by the Oracle Applications alias OA_HTML.

See Table 16–2, " Constraints on the HTML Template" on page 16-5 and Table 16–3, " Correspondence of HTML to JSP Files" on page 16-7 for a listing of the files required to be located in the HTML directory, on all platforms.

In addition, the file `czAll.js`, which contains the JavaScript controls for the DHTML user interface of the runtime Oracle Configurator, must be located in the HTML directory.

If you are implementing Oracle Configurator in a language other than English, you may need to translate the language-specific NLS files:

- `czBlafTemplate.jsp`
- `czFraTemplate.jsp`
- `czFormTemplate.jsp`
- `czIFrame.jsp`
- `czLeft.jsp`
- `czRight.jsp`

12.1.4 Files for the Media Directory

The image files in the Media directory are used by the runtime Oracle Configurator embedded in your host application to decorate the DHTML Oracle Configurator window, and also to represent application logic state in the DHTML controls in the Oracle Configurator window.

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, since this file format is not compatible with web browsers.

This affects you if the User Interface that you define in Oracle Configurator Developer includes any BMP files.

The Oracle Configurator window can use GIF, JPG, and other formats compatible with web browsers.

13

Pricing and ATP in Oracle Configurator

How Oracle Configurator handles pricing and ATP (Available To Promise) 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.

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.

13.1 Pricing in a Runtime Oracle Configurator

For a list of hosting applications that support Oracle Configurator, see the *Oracle Configurator Release Notes*.

The Java Applet interface for the runtime Oracle Configurator presents list prices for all selectable options, selling prices (for example, discounts) for all selected options, and a total price for the configuration as a whole. The Dynamic HTML in a browser interface presents *either* list prices for all selectable options, *or* selling prices for all selected options, and enables you to add a total price.

When the host application is part of Oracle Applications, such as Order Management, then pricing 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 in order to determine pricing parameters. The host application must implement the Oracle Configurator pricing interface package, as described in Section 13.3.2 on page 13-6. 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

For more information about the Pricing Callback Interface, see Section 13.3.2 on page 13-6.

13.1.1 Pricing Interaction in the User Interfaces

The interaction of the end user with pricing varies depending on the Oracle Configurator user interface.

13.1.1.1 Pricing Interaction in the Java Applet User Interface

When the runtime Oracle Configurator is initialized, it displays the list prices for the options appearing on the first screen page, in the Options Selection region and in the Selected Items or Summary region.

When the end user clicks the Update Prices button, the runtime Oracle Configurator calls the pricing package to get the selling prices for all selected options and the total price for the configuration. The selling prices are displayed in the Selected Items and Options Selection windows.

13.1.1.2 Pricing Interaction in the DHTML User Interface

When the end user clicks the Summary button in the DHTML UI, the runtime Oracle Configurator calls the pricing package to get the selling prices for all selected options and the total price for the configuration. The selling prices are then displayed on the Summary screen and on the Configuration screen.

13.1.2 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, since 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.

13.2 Runtime Oracle Configurator Pricing and ATP Architecture

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.

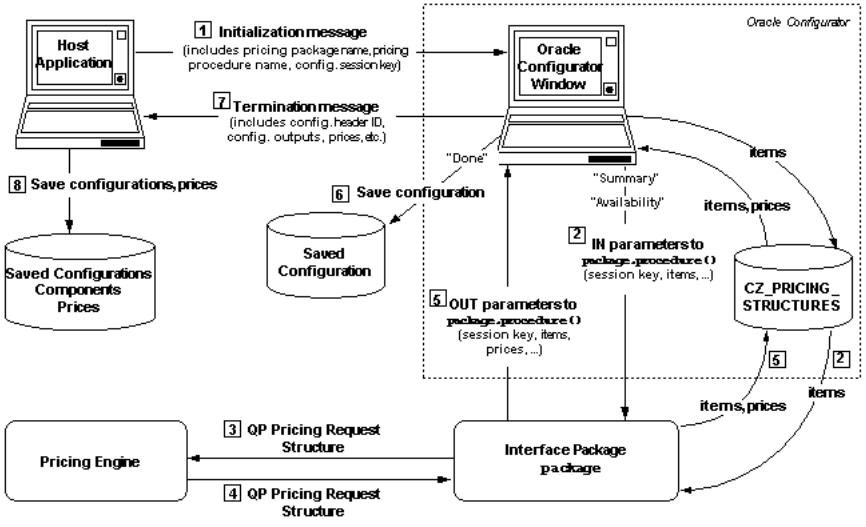
The CZ_PRICING_STRUCTURES table does not support pricing rules based on the fact that items belong to the same instance. Pricing is done per component instance.

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.

Figure 13–1, "Runtime Oracle Configurator Pricing Architecture", illustrates this architecture. Illustrated steps 2 through 5 can be repeated many times. Note that in Figure 13–1, all of the database symbols refer to the same instance of the CZ schema.

Figure 13–1 Runtime Oracle Configurator Pricing Architecture



See the Section 13.3.2 on page 13-6 for details about the pricing interface package, and see Chapter 9, "Session Initialization"and Chapter 10, "Session Termination"for details about the initialization and termination messages for a runtime Oracle Configurator session.

13.3 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 (e.g., through the coding of the "Configure" button) to post the XML initialization message to the OC Servlet, 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 Section 13.3.1, "Database Compatibility" on page 13-5.
2. Write your own PL/SQL pricing or ATP procedures, using the OC interfaces. See Section 13.3, "Integration of Pricing and ATP with Oracle Configurator" on page 13-5 for details.

   See Section D.1, "Pricing and ATP Callback Procedures" on page D-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 procedure to use, and the type of pricing to perform.

   See Section 13.3.4, "Initialization Parameters" on page 13-14 for an example. See Section 9.5.5, "Pricing Parameters" on page 9-13 and Section 9.5.6, "ATP Parameters" on page 9-14 for explanation of the parameters.

13.3.1 Database Compatibility

Oracle Configurator works natively with Oracle Applications Release 11*i*, using the Oracle8*i* Enterprise Edition (Release 8.1.x) database.

In order to obtain pricing data from an Oracle8 Enterprise Edition (Release 8.0.x) database, as used with Oracle Applications 10.7 / 11.0, you must run a concurrent program. See the Section B.4, "Populate and Refresh Configuration Models Concurrent Programs" on page B-11.

There are several likely scenarios for pricing and ATP integration. These are described in the following table:

To Integrate with...	You would...
Oracle Applications Release 11*i* database	You write your own callback procedures (which can call the QP Advanced Pricing engine). To import BOM data to the Oracle Configurator schema tables, you run concurrent programs in the Oracle Bills Of Material application. To export orders to Order Management (Oracle Applications Release 11*i*), 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.

13.3.2 The Pricing Callback Interface

The pricing callback interface package provides interfaces for two distinct procedures:

- Price Single Item
- Price Multiple Items
- Price Multiple Items for MLS

The Price Single Item procedure returns price information for a single item. Table 13–1 describes the parameters for the Price Single Item procedure.

Table 13–1 Price Single Item Procedure Parameters

Parameter	In/Out	Type	Required	Note
`configurator_session_key`	In	Varchar2	Required	Limit of 50 characters
`price_type`	In	Varchar2	Required	Values are: `LIST` or `SELLING`

Table 13–1 (Cont.) Price Single Item Procedure Parameters

Parameter	In/Out	Type	Required	Note
ps_node_id	In	Number	Conditionally Required	Either ps_node_id or item_key is required.
item_key	In	Varchar2	Conditionally Required	Either ps_node_id or item_key is required.[1]
quantity	In	Number	Required	
uom_code	In	Varchar2	Required	
list_price	Out	Number	n/a	
selling_price	Out	Number	n/a	This is the unit selling price. OC automatically multiplies this price by quantity to produce the total price (extension) for the item.
msg_data	Out	Varchar2	n/a	

[1] If the model was imported from an Oracle BOM, then this key is COMPONENT_CODE:EXPLOSION_TYPE:ORGANIZATION_ID:TOP_ITEM_ID.

The Price Multiple Items procedure returns price information for a group of items. Table 13–2 describes the parameters for the Price Multiple Items procedure.

Table 13–2 Price Multiple Items Procedure Parameters

Parameter	In/Out	Type	Required	Note
configurator_session_key	In	Varchar2	Required	
price_type	In	Varchar2	Required	Values are: LIST or SELLING
config_total_price	Out	Number	n/a	

The Price Multiple Items MLS procedure returns price information for a group of items. Table 13–3 describes the parameters for the Price Multiple Items MLS procedure.

Table 13–3 Price Multiple Items MLS Procedure Parameters

Parameter	In/Out	Type	Required	Note
configurator_session_key	In	Varchar2(50)	Required	
price_type	In	Varchar2	Required	Values are: LIST, SELLING or BOTH
config_total_price	Out	Number	Required	
currency_code	Out	Varchar2	Required	
thousands_separator	Out	Varchar2	Required	
decimal_separator	Out	Varchar2	Required	
positive_currency_format	Out	Varchar2	Required	
negative_currency_format	Out	Varchar2	Required	
precision	Out	Varchar2	Required	

The parameters of the interface are passed by positional notation, so you can name the parameters as desired, as long as you retain the positionality specified in Table 13–1, Table 13–2, and Table 13–3.

13.3.2.1 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 table is described in Table 13–4 on page 13-8. For details on Oracle Configurator tables, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

Table 13–4 CZ_PRICING_STRUCTURES Database 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.

Table 13–4 (Cont.) CZ_PRICING_STRUCTURES Database Table

Column Name	Data Type	Null?	Description
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 1[1] if ITEM_KEY is ORIG_SYS_REF. Set to 2[2] 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
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. Limit of 15 digits
PARENT_CONFIG_ITEM_ID	Number		Together with CONFIG_ITEM_ID, this establishes the full hierarchy of the configuration when there are multiple instantiations.

[1] Value of CZ_PRC_CALLBACK_UTIL.G_ITEM_KEY_BOM_NODE.

[2] Value of CZ_PRC_CALLBACK_UTIL.G_ITEM_KEY_PS_NODE.

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. 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, OC deletes the pricing records from the CZ_PRICING_STRUCTURES table.

If your host application needs to 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.

13.3.2.2 Examples of the Pricing Callback Interface

Example 13–1 on page 13-10 shows a possible implementation of the callback interface for both single and multiple-item pricing procedures.

Example 13–3 on page 13-14 shows how you would specify pricing parameters in your initialization message.

Example D–1 on page D-2 provides a minimal example of the use of the callback interface.

Example 13–1 Pricing Callback Interface

```
PACKAGE CZ_PRICE_TEST AUTHID CURRENT_USER AS

 PROCEDURE price_single_item (configurator_session_key IN VARCHAR2,
                              price_type IN VARCHAR2,
                              ps_node_id IN NUMBER,
                              item_key IN VARCHAR2,
                              quantity IN NUMBER,
                              uom_code IN VARCHAR2,
                              list_price OUT NUMBER,
                              selling_price OUT NUMBER,
                              msg_data OUT VARCHAR2);

  PROCEDURE price_multiple_items (p_configurator_session_key IN VARCHAR2,
                                  p_price_type IN VARCHAR2,
                                  p_total_price OUT NUMBER);

END;
```

13.3.3 The ATP Callback Interface

The "Get ATP Dates" procedure returns availability dates for all selected options and for the configuration as a whole. Table 13–5 describes the parameters for the Get ATP Dates procedure.

Table 13–5 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	
customer_site_id	In	Number	Conditionally Required	
requested_date	In	Date	Required	Defaults to value of SYSDATE.
ship_to_group_date	Out	Date	n/a	

The parameters of the interface are passed by positional notation, so you can name the parameters as desired, as long as you retain the positionality specified in Table 13–5.

13.3.3.1 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. This table is described in Table 13–6 on page 13-11. For details on Oracle Configurator tables, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

Table 13–6 CZ_ATP_REQUESTS Database Table

Column Name	Data Type	Null?	Description
ATP_REQUEST_ID	Number	Not Null	Primary key.
CONFIGURATOR_SESSION_KEY	Varchar2	Not Null	Limit of 50 characters. Identifies a configurator session. Only one configuration can be handled in the session.

Table 13–6 (Cont.) CZ_ATP_REQUESTS Database Table

Column Name	Data Type	Null?	Description
SEQ_NO	Number	Not Null	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		Limit of 9 digits. Set to 1[1] if ITEM_KEY is ORIG_SYS_REF. Set to 2[2] if ITEM_KEY is PS_NODE_ID.
QUANTITY	Number		Limit of 9 digits. Item quantity
UOM_CODE	Varchar2		Limit of 3 characters. UOM code
MSG_DATA	Varchar2		Limit of 2000 characters. Message text filled in by your host application.
INV_ORG_ID	Number		Limit of 9 digits.
SHIP_TO_DATE	Date		
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. Limit of 15 digits

Table 13–6 (Cont.) CZ_ATP_REQUESTS Database Table

Column Name	Data Type	Null?	Description
PARENT_CONFIG_ ITEM_ID	Number		Together with CONFIG_ITEM_ID, this establishes the full hierarchy of the configuration when there are multiple instantiations. Limit of 15 digits.

[1] Value of CZ_PRC_CALLBACK_UTIL.G_ITEM_KEY_BOM_NODE.

[2] Value of CZ_PRC_CALLBACK_UTIL.G_ITEM_KEY_PS_NODE.

Your ATP package must retrieve the items from this 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 needs to 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.

13.3.3.2 Examples of the ATP Callback Interface

Example 13–2 on page 13-13 shows an implementation of the callback interface for ATP procedures.

Example 13–3 on page 13-14 shows how you would specify ATP parameters in your initialization message.

Example D–3, "Example of Callback ATP Procedure" on page D-3 provides an example in context.

Example 13–2 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 DATE);

END cz_atp_callback;
```

13.3.4 Initialization Parameters

Example 13–3 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 11*i*. See Section 9.5.5, "Pricing Parameters" on page 9-13 and Section 9.5.6, "ATP Parameters" on page 9-14.

Example 13–3 Initialization message using 11i pricing and ATP parameters

```
<html>
<head>
<title>Project Test</title>
<script language="javascript">
function init() {
  document.test1.submit();
}
</script>
</head>

<body onload="init();">
<form
action="http://www.mysite.com:10130/servlets/oracle.apps.cz.servlet.UiServlet"
method="post" id="test1" name="test1"><input type="hidden" name="XMLmsg"
value='<initialize>
<param name="alt_database_name">jdbc:oracle:thin:@server01:1521:vis11</param>
<param name="user">apps</param>
<param name="pwd">apps</param>
<param name="ui_type">DHTML</param>
<param name="gwyuid">APPLSYSPUB/PUB</param>
<param name="fndnam">APPS</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="price_single_item_proc">price_single_item</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>

</initialize>'></form>

<br>Loading configurator...

</body>
</html>
```

In order to obtain the final prices calculated by your pricing package, you need to specify a value of `full` for the initialization parameter terminate_msg_behavior. 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.

13.4 Controlling Pricing in the Runtime Oracle Configurator

You can control whether and how pricing and ATP information is presented in the runtime Oracle Configurator.

- You can control which types of prices appear.
- You can control when pricing data is obtained from the database.

To control both of these aspects of pricing behavior in the runtime Oracle Configurator, you must:

1. In Oracle Internet Application Server, set the value of the `cz.activemodel` and `cz.activemodel.lazyloadlistprice` properties of the OC Servlet.

   See the *Oracle Configurator Installation Guide* for more details about setting OC Servlet properties.

2. In Oracle Configurator Developer, choose options in the Pricing Display attribute of the User Interface for your Model.

   In order to take effect, these settings require that the properties mentioned in Step 1 be set. The Pricing Display settings in Oracle Configurator Developer modify, but cannot override, the effect of the properties of the OC Servlet.

See the *Oracle Configurator Developer User's Guide* for more details about setting the Pricing Display attribute with Oracle Configurator Developer, and about differences in pricing display behavior between the Java Applet and Dynamic HTML in a browser.

3. If you are creating a custom application, you must also set the appropriate parameters in the initialization message that is posted to the OC Servlet.

   See the Section 13.3.2 on page 13-6 for details about the pricing interface package, and see Chapter 9, "Session Initialization"and Chapter 10, "Session Termination"for details about the initialization and termination messages for pricing and ATP.

13.4.1 Displaying Price Types

The following sections explain how to control the type of prices to be displayed to the end user.

13.4.1.1 Setting the OC Servlet Property for Price Types

In order to control the type of prices to be displayed, you must first set the `cz.activemodel` property of the OC Servlet. (See the *Oracle Configurator Installation Guide* for details about setting OC Servlet properties.)

The syntax for this setting is:

```
cz.activemodel=/switch|
```

Table 13–7, " Switch Effects" lists the `switch` and its effect:

Table 13–7 Switch Effects

Switch	Effect
lp	Fetch List Prices from the database.
dp	Fetch Discounted Prices from the database.
atp	Fetch ATP data from the database.
nolp	Do not fetch List Prices from the database.
nodp	Do not fetch Discounted Prices from the database.
noatp	Do not fetch ATP data from the database.

You can set more than one switch. The syntax for this setting is:

```
cz.activemodel=/switch1|/switch2|
```

Note that each switch is separated by the pipe character (|), and that the pipe character is required at the end of the property setting.

If you set contradictory switches, then only the first switch is respected. Example:

```
cz.activemodel=/lp|/nolp|
```

```
The default pricing behavior for the OC Servlet is that all price fetching is
turned off. You can produce this behavior by omitting the cz.activemodel
property from the OC Servlet's configuration files.
```

If one of the switches that disable fetching has been set, the end user will receive an error message when pricing is requested in the runtime Oracle Configurator.

Be aware that in Dynamic HTML in a browser, the display of columns for Discount Price and Extended Price is controlled by the initialization message. If you pass pricing parameters, these columns will be displayed.

Examples of Setting Pricing Switches

Table 13–8, " Examples and Effect of Pricing Switches" lists examples and the effect of pricing switches:

Table 13–8 Examples and Effect of Pricing Switches

Setting	Effect
`cz.activemodel=/lp|`	List Prices will be fetched and displayed.
`cz.activemodel=/nolp|`	List Pricing is turned off. List Prices will not be fetched or displayed.
`cz.activemodel=/lp|/dp|`	List Prices and Discounted Prices will both be fetched and displayed.
`cz.activemodel=/nolp|/nodp|`	No List Prices or Discounted Prices will be fetched.

13.4.1.2 Setting the Item Price Display in Oracle Configurator Developer

If you have enabled the appropriate price fetching in the OC Servlet, then you can modify it in the User Interface for your Model, using Oracle Configurator Developer.

In Oracle Configurator Developer, go to the User Interface module for your Model, and choose an option for the Item Price Display setting of the Pricing Display attribute. Table 13–9, " Item Price Option Effects" lists the options and the effects:

Table 13–9 Item Price Option Effects

Option	Effect
List Prices	Display the unit list prices for each item.
Selling Prices	Display the unit selling price (discounted price) of all selected items.
None	Do not display any List Prices or Selling Prices in the UI.

For a DHTML User Interface, you can choose either List Prices or Selling Prices, but not both.

See the *Oracle Configurator Developer User's Guide* for more details about setting the Pricing Display attribute with Oracle Configurator Developer.

These options are overridden by the settings for the OC Servlet property `cz.activemodel`. For instance, if `cz.activemodel` is set to `/nolp|`, then you cannot turn it on by choosing the List Price option in Configurator Developer.

13.4.2 Updating Price Data

The fetching and display of pricing information may impose a significant additional load on the performance of Oracle Configurator. You can improve performance by fetching pricing data only when it is needed for the end users of your application.

13.4.2.1 Setting the OC Servlet Property for Price Updating

In order to control when list price data is fetched, you must first set the `cz.activemodel.lazyloadlistprice` property of the OC Servlet. See the *Oracle Configurator Installation Guide* for details about setting OC Servlet properties.

The syntax for this setting is:

```
cz.activemodel.lazyloadlistprice=[true|false]
```

If you set this property to `true`, then list pricing data is only fetched for the items on the page that is currently being displayed. When the runtime Oracle Configurator is initialized, only the list prices for the items on the first page are fetched and displayed. Thereafter, list prices are displayed for each new page that the end user navigates to.

If you set this property to `false`, then list pricing data is fetched for the entire configuration Model.

The default value for this property is `true`. This property controls the fetching of list prices only, not of selling prices or total price.

13.4.2.2 Setting the Price Update in Oracle Configurator Developer

If you have enabled the appropriate price updating in the OC Servlet, then you can modify it in the User Interface for your Model, using Oracle Configurator Developer. While this method can affect both list and selling prices, it is useful mainly as a way of controlling selling price updating, since you can control list price updating as described in Section 13.4.2.1, "Setting the OC Servlet Property for Price Updating" on page 13-18.

In Oracle Configurator Developer, go to the User Interface module for your Model, and choose an option for the Price Update setting of the Pricing Display attribute. Table 13–10, " List Price Property Settings" lists the Option and the corresponding effect.

See the *Oracle Configurator Developer User's Guide* for more details about setting the Pricing Display attribute with Oracle Configurator Developer.

These options are overridden by the settings for the OC Servlet property `cz.activemodel.lazyloadlistprice`. For instance, if `cz.activemodel.lazyloadlistprice` is set to `true`, then the Always option in Configurator Developer will be ignored. If `cz.activemodel.lazyloadlistprice` is set to `false` in the OC Servlet, then these options in Configurator Developer have no effect on list price updating, since list prices will be displayed for all items in the Model.

13.4.3 Examples of Controlling Pricing

This section illustrates how the various settings that control pricing can be used together. See the rest of for an explanation of these choices.

13.4.3.1 Example: List Prices Only

Table 13–10, " List Price Property Settings" illustrates the values that you should probably choose— subject to the specific nature of your application—if you want to display only list prices.

Table 13–10 List Price Property Settings

Property or Setting	Value
`cz.activemodel`	`/lp|/nodp|`

Table 13–10 (Cont.) List Price Property Settings

Property or Setting	Value
Price Display	List Prices
`cz.activemodel.lazyloadlistprice`	`true` (probably, depending on application)
Price Update	any setting

13.4.3.2 Example: Selling Prices Only

Table 13–11, " Selling Price Property Settings" illustrates the values that you should probably choose— subject to the specific nature of your application—if you want to display only selling prices.

Table 13–11 Selling Price Property Settings

Property or Setting	Value
`cz.activemodel`	`/nolp|/dp|`
Price Display	Selling Prices
`cz.activemodel.lazyloadlistprice`	`true` (or omit altogether)
Price Update	any setting

14

Multiple Language Support

Each Oracle Applications installation has one base language but can have additional installed languages. Multiple Language Support (MLS) enables you to create configuration models and one or more user interfaces in your base language, and then display the Model in any language in which you do business. For a list of all languages that Oracle Applications supports, see the *Oracle 8i National Language Support Guide*.

Descriptions for BOM items are entered in Oracle Applications. You enter descriptions for nodes created in Configurator Developer using PL/SQL or a custom translation facility. See Section 14.5, "Modify Description Translation Supported by MLS" on page 14-4 for sample PL/SQL translations. Item names cannot be translated.

Oracle Configurator Developer menus, prompts and field names appear in English only. All information entered in Oracle Configurator Developer is implemented with the language that is installed on the developer's machine.

14.1 Data Import

When defining an Item in Oracle Inventory, an Oracle Applications user can enter an alternate description for the Item in each language installed on the system. (You cannot enter translations for Item names in Oracle Applications.) For example, the base language of your Oracle Applications installation is English, but French, Spanish, and German are installed. When creating a new Item, an Oracle Inventory user enters the Item name and description in English, then uses the Translation window to enter alternate descriptions in French, Spanish, and German. (For more information, see Creating Translations for a Record in the *Oracle Applications User's Guide*.) All translatable strings for BOMs created in Oracle Applications are available in the MTL_SYSTEM_ITEMS_TL table.

When you import a BOM Model into the CZ schema, the Item descriptions are also imported into the CZ schema, including the base language and any alternate translations.

Before importing a BOM, be sure that all Items defined in Oracle Inventory contain descriptions. If an Item does not have a description, both the name and description columns in the DHTML Summary screen may be blank at runtime. See the *Oracle Configurator Installation Guide* for information about the MLS Servlet property option to hide the Item name.

Warning: The Populate/Refresh Configuration Models concurrent programs import any new or modified descriptions from Oracle Applications and overwrite any BOM Item descriptions that you may have modified in the CZ schema. Therefore, it is recommended that you enter and modify the translated Item descriptions only in Oracle Inventory.

Import extracts all strings (translated or not) from MTL_SYSTEM_ITEMS_TL for all languages installed on the import target machine. Import populates CZ_LOCALIZED_TEXTS with MTL_SYSTEM_ITEMS_TL.DESCRIPTION. If you want to publish the UI and display information in all installed languages, you must manually translate the description of each non-BOM node (remember that all translated descriptions are imported with the BOM). You can translate non-BOM descriptions using either SQL*Plus or a translation facility that you create.

It is recommended that you review the predefined tooltip text and all Configurator Developer validation, warning, and error messages to ensure that the translated version of each message exists and is correct. Then, modify the information using SQL*Plus or a custom translation facility as required. Tooltip text and any messages created in Configurator Developer are stored in the CZ_LOCALIZED_TEXTS table. All predefined Configurator Developer messages are stored in the FND_NEW_MESSAGES table. See Section 14.5, "Modify Description Translation Supported by MLS" on page 14-4.

During the Populate/Refresh Configuration Models concurrent programs, if SOURCE_LANG in the source database is different from LANGUAGE in the source database and if the SOURCE_LANG and LANGUAGE are the same in the target database, then the descriptions are not imported. They are marked as 'Rejected' with a status of 'R'.

Warning: When importing from a remote server, the list of installed languages on the source database must be the same as the list of installed languages on the target database.

14.1.1 New Models

When importing a new BOM Model, the Oracle Configurator import procedures import all available translations for each item in the BOM.

14.1.2 Existing Models

Existing configuration models must be refreshed so that the CZ_LOCALIZED_ TEXTS table is populated with the new or modified translations that have been entered in Oracle Inventory for the BOM Models.

14.2 Creating User Interfaces

When generating a new UI for use with multiple languages, you must create your UI captions using the *description* of each node. Changing the name corrupts your configuration model. The Languages applicability parameter determines in which languages a publication is available at runtime.

When publishing the Model and UI, you specify in which languages the publication is available using the Languages applicability parameter. All language translations defined for the model are exported when you create the publication. When the publication is accessed at runtime, the language specified by the host application determines which language to display in the UI. See the *Oracle Configurator Developer User's Guide* for more information about publishing and MLS.

If you are using the Java applet UI, then you must use Oracle JInitiator version 1.1.8.7. This version must be specified in your spx.ini file. See the *Oracle Configurator Installation Guide* for details.

14.3 Precedence of Language Settings

When determining which language to display in the Configurator window, the runtime Oracle Configurator searches the following sources, in this order:

1. The Language property of the AppsContext object. This property's value is derived from the settings in the ICX Session Ticket. If no ICX Session Ticket is

found, then the default language associated with the AppsContext object is used.

2. The base language of the Oracle Applications database.

14.4 Installed Languages

If you are publishing in a two-server environment (such as a development and a production database), the base language and the set of installed languages on both Oracle Applications servers must be exactly the same.

14.5 Modify Description Translation Supported by MLS

See the *Oracle Configurator Developer User's Guide* for additional information about MLS.

There are several tables that can be used when writing queries to update the descriptions for a specified source language and language.

CZ_DEVL_PROJECTS.INTL_TEXT_ID - this is the international text identifier for the specific project

CZ_PS_NODES.VIOLATION_TEXT_ID - this is the violation message when a Resource is over consumed

CZ_RULES.REASON_ID - this is the violation message for a rule

CZ_RULES.UNSATISFIED_MSG_ID - this is the message when the rule is not satisfied

CZ_UI_NODES.TOOL_TIP_ID - this is the INTL_TEXT record that holds the text for a 'tool tip' describing the UI component

The following sections are suggested PL/SQL queries that can be run to update appropriate nodes with translations.

14.5.1 Update the PS Node Description With Translations

1. First determine the development project's ID:

```
SQL> select devl_project_id, name
  2  from cz_devl_projects
  3  where name like '%Sen%'
```

DEVL_PROJECT_ID	NAME
2020	Sentinel Custom Desktop (204 137)

2. Using the DEVL_PROJECT_ID in the following query determines whether PS_NODE_IDs have INTL_TEXT records. If a PS node was created without a description, it would not have an INTL_TEXT_ID.

```
SQL> ps_node_id, name, intl_text_id
  2  from cz_ps_nodes
  3  where devl_project_id = 2020
```

PS_NODE_ID	NAME	INTL_TEXT_ID
2601	Total RAM Available	10781
3960	Software-Web Browser	

3. Add a description for Software-Web Browser and then run the query to get the INTL_TEXT_ID.

```
SQL> ps_node_id, name, intl_text_id
  2  from cz_ps_nodes
  3  where devl_project_id = 2020
```

PS_NODE_ID	NAME	INTL_TEXT_ID
2601	Total RAM Available	10781
3960	Software-Web Browser	**10959**

4. Using the appropriate DEVL_PROJECT_ID, the following query returns those PS_NODES_ID that have INTL_TEXT_IDs, and their descriptions in LOCALIZED_STR.

```
SQL> column name format a20
column localized_str format a40
select   a.ps_node_id, a.name, a.intl_text_id, b.source_lang, b.language,
b.localized_str
from cz_ps_nodes a, cz_localized_texts b
where a.intl_text_id = b.intl_text_id
and devl_project_id = 2020
 order by a.ps_node_id
```

PS_NODE_ID	NAME	INTL_TEXT_ID	SOUR	LANG	LOCALIZED_STR
2916	CM56560	10959	US	AR	Software-Web Browser
2916	CM56560	**10959**	US	F	Software-Web Brpwser
2916	CM56560	10959	US	US	Software-Web Browser
2916	CM56560	10959	US	JA	Software-Web Browser

5. Use the INTL_TEXT_ID in the following SQL statement to update the description with the French translation. SOURCE_LANG must be updated when translating to establish that the translation has been accomplished

```
SQL> update cz_localized_texts
set localized_str = 'Logiciel-Web Browser (Description)'
source_lang = 'F'
where intl_text_id = 10959
and language = 'F'
```

6. Verify the new French translation and commit:

```
SQL> select intl_text_id, language, localized_str, source_lang
from cz_localized_texts
where intl_text_id = 10959
commit
```

INTL_TEXT_ID	SOURCE_LANG	LANG	LOCALIZED_STR
10959	US	AR	Software-Web Browser
10959	F	F	Logiciel-Web Browser (Description)
10959	US	US	Software-Web Browser
10959	US	JA	Software-Web Browser

14.5.2 Updating the UI Node's Label Text with Translations

1. First you must determine the development project's ID:

```
SQL> select devl_project_id, name
```

```
2  from cz_devl_projects
3  where name like '%Sen%'
```

DEVL_PROJECT_ID	NAME
2020	Sentinel Custom Desktop (204 137)

2. Get the UI_DEF_ID:

```
SQL> select name, ui_def_id
from cz_ui_defs
where devl_project_id = 2020
```

UI_DEF_ID	NAME
2340	Sentinel Custom Desktop (204 137) User Interface

3. Use the UI_DEF_ID in the following SQL statement to retrieve the UI node's INTL_TEXT_IDs.

```
SQL> column "UI Node Name" format a25
column localized_str format a40
select   a.caption_id "INTL_TEXT_ID", ui_node_id, a.name "UI Node Name",
          b.source_lang, b.language, b.localized_str
  from    cz_ui_nodes a, cz_localized_texts b
 where    a.caption_id=b.intl_text_id and a.ui_def_id = 2340
   and    b.localized_str like '%Web Browser%'
```

INTL_TEXT_ID	UI Node Name	ui_node_id	SOUR	Lang	LOCALIZED_STR
11459	Software-Web Browser	68940	US	JA	Software-Web Browser
11459	Software-Web Browser	68940	US	AR	Software-Web Browser
11459	Software-Web Browser	68940	US	F	Software-Web Browser
11459	Software-Web Browser	68940	US	US	Software-Web Browser
11460	Text - 70300	68942	US	JA	Software-Web Browser
11460	Text - 70300	68942	US	AR	Software-Web Browser

INTL_TEXT_ID	UI Node Name	ui_node_id	SOUR	Lang	LOCALIZED_STR
11460	Text - 70300	68942	US	F	Software-Web Browser
11460	Text - 70300	68942	US	US	Software-Web Browser

4. Use the UI node's INTL_TEXT_ID in the following SQL statement to update the UI node's label text with the French translation.

```
SQL> update cz_localized_texts
set localized_str = 'Logiciel - Web Browser (Label Text)', source_lang = 'F'
where intl_text_id in (11460) and language = 'F'
```

5. Update the CZ_UI_NODES.MODIFIED FLAGS to non-zero value. This ensures that the PS_NODE descriptions do not overwrite the UI node's label text during refresh.

```
SQL> update cz_ui_nodes
set modified_flags = 1
where caption_ID = 11460
```

6. Verify changes and commit.

```
SQL> select   a.intl_text_id, b.ui_node_id, b.name "UI Node Name", a.source_
lang, a.language, a.localized_str, b.modified_flags
  from cz_localized_texts a, cz_ui_nodes b
 where a.intl_text_id = b.caption_id and a.intl_text_id in (11460)
```

INTL_TEXT_ID	ui_node_id	UI_NODE_NAME	SOUR	Lang	LOCALIZED_STR	Modifie
11460	68942	Text - 70300	US	AR	Software-Web Browser	
11460	68942	Text - 70300	F	F	Logiciel - Web Browser	Label Text
11460	68942	Text - 70300	US	US	Software-Web Browser	

```
commit
```

Part IV

Configuration Model

Part IV presents information that enables you to extend a BOM Model's structure, rules, and UI to reflect your business needs and integrate with a hosting application as described in Section 1.4, "Model Development Tasks" on page 1-5. Part IV contains the following chapters:

15

Controlling the Development Environment

Oracle Configurator Developer is both a development and maintenance environment used to create and modify configuration models and custom Oracle runtime configurator windows.

15.1 Oracle Configurator Developer

Oracle Configurator Developer is a database-intensive application and requires a fast connection to your Oracle Configurator schema. We recommend the use of a local area network (LAN). A wide area network (WAN) configuration will only be practical if the network bandwidth is high enough. However, in many situations, it will be necessary to run Configurator Developer remotely using a Windows Terminal Server (WTS) server located on the same server as the database server.

For example, customers trying to use an application service provider (ASP) will have to use a form of WTS. For more information about using a WTS, see the *Oracle Configurator Installation Guide*.

For more background on the relationship of Oracle Configurator Developer to the Oracle Configurator architecture, see Chapter 2, "Configurator Architecture".

15.1.1 Profile Options

Profile Options affect the way Oracle Applications look and behave. This behavior includes the way in which a host application interacts with the runtime Oracle Configurator. A System Administrator can set user profile values at the site, application, responsibility and user levels. See *Oracle Applications System Administrator's Guide* for more information.

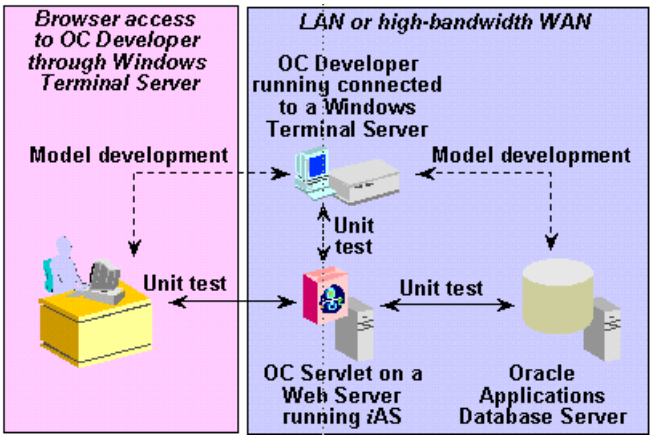
For a listing of Oracle Configurator Profile Options, see the *Oracle Configurator Installation Guide*.

15.1.2 Model Development

Using Oracle Configurator Developer that is connected by a LAN, WAN or a WTS to the database server, 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 Figure 15–1, "Developer Environment".

After making modifications to the Model, access to the runtime Oracle Configurator is gained through the OC Developer's **Test** button, which invokes the OC Servlet. The OC Servlet commits unit-testing configuration data to the database, after the developer closes the Configurator window. This is shown as the Unit test scenario in Figure 15–1, "Developer Environment".

Figure 15–1 Developer Environment



15.1.3 Runtime Testing

All Oracle Configurator runtime database commits are through OC Servlet. When the end user closed the Configurator window, the resulting configuration data is saved directly to the database.

In order to test the configuration model, there are certain objects that must be in place:

- InitServletURL parameter must be set in the `spx.ini` file.
- Class path for Functional Companions must be set. In order for a Functional Companion to work, the Java class that implements it must be in the class path

of the OC Servlet. Edit the appropriate Apache configuration file (such as `jserv.properties`). For example, if the Java classes for your Functional Companions are in a Java archive file named `fc.jar`, then you would add a line to `jserv.properties`:

```
wrapper.classpath=jserv_install/servlets/fc.jar
```

See the *Oracle Configurator Installation Guide* for more information.

- OC Servlet must be restarted if you add or modify the Java class for a Functional Companion.
- Open a new configuration session in a new browser window by clicking the **Test** button in order 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:

```
http://host:port/configurator/oracle.apps.cz.servlet.UiServlet?test=version
```

15.2 Spx.ini

The `spx.ini` file contains parameters that determine connectivity and product behavior for the development and testing of a runtime Oracle Configurator.

During installation of Oracle Configurator Developer, a default `spx.ini` file is copied to the winnt directory (for Windows NT machines) or the Windows directory (for Windows 95/98 machines). If the installation procedure encounters an existing `spx.ini` file, it renames that file spx_ini.bak, so that you do not lose the edits you made when you upgrade or reinstall Oracle Configurator Developer. To restore the edits you made to `spx.ini`, examine the new `spx.ini` file and copy any new entries in this file to the existing `spx_ini.bak` back to `spx.ini` as new functionality may have been introduced.

See Section 15.3, "Parameters in spx.ini" on page 15-5 for details about specific parameters.

Example 15–1 is the `spx.ini` file that is installed with Oracle Configurator.

Example 15–1 Default Installed spx.ini

```
[Merlin]
;In order to use Developer please uncomment (i.e. remove the ';') from
```

```
;the sample lines for each odbc dsn and replace each word
;<odbc-dsn> or <host> etc. with a valid dsn, hostname etc.. Note that
;characters such as '[' and '@' should not be replaced and need to be
;in the line after you finish editing it

[DSN]
;odbc-dsn-1
;odbc-dsn-2

;[odbc-dsn-1]
;DBOwner=db-owner
;JdbcUrl=jdbc:oracle:thin:@host:portnumber:SID
;Launch=1
;InitServletURL=http://www.myhost.com:port/configurator/oracle.apps.cz.servlet.U
iServlet
;jinitVersion=1.1.8.7
;jinitClassID = clsid:ff348b6e-fd21-11d4-a3f0-00c04fa32518
;gwyuid=applsyspub
;gwypass=pub

[odbc-dsn-2]
;DBOwner=db-owner
;JdbcUrl=jdbc:oracle:thin:@host:portnumber:SID

[Design Chart]
DEF=M
SEC=X
```

Example 15–2 shows a modified `spx.ini` file. The myserv instance uses the [Merlin] default values for InitServletURL and InitCodeBaseUrl as those values are not specified in the [dsn]. Similarly, czapp instance uses the [Merlin] default values for Launch, InitServletURL and InitCodeBaseUrl. See Section 15.3.1, "[Merlin]" on page 15-6 for more information.

Example 15–2 Modified spx.ini File

```
[Merlin]
DBOwner=apps
JdbcUrl=jdbc:oracle:thin:@host:portnumber:sid
Launch=1

InitServletURL=http://www.myhost.com:port/configurator/oracle.apps.cz.servlet.Ui
Servlet
InitCodeBaseUrl=http://app882sun.us.oracle.com:10140/applet
```

```
[DSN]
sales
myserv
training

[sales]
DBOwner=apps
JdbcUrl=jdbc:oracle:thin:@host:portnumber:sales
gwyuid=APPLSYSPUB
gwypass=PUB
APPLID=660
RESPID=21623
Launch=1

[myserv]
DBOwner=apps
JdbcUrl=jdbc:oracle:thin:@host:portnumber:myserv
gwyuid=APPLSYSPUB
gwypass=PUB
APPLID=660
RESPID=21623

[training]
DBOwner=apps
JdbcUrl=jdbc:oracle:thin:@host:portnumber:training
gwyuid=APPLSYSPUB
gwypass=PUB
APPLID=660
RESPID=21623
Launch=2
InitServletURL=http://www.myhost.com:port/configurator/oracle.apps.cz.servlet.Ui
Servlet
InitCodeBaseUrl=http://app882sun.us.oracle.com:10140/applet

[Design Chart]
DEF=M, N, A, T
SEC=X, Y, Z
```

15.3 Parameters in spx.ini

The `spx.ini` file sets the DBOwner and other parameters for running:

- Oracle Configurator Developer

- A test instance of a runtime Oracle Configurator from within Oracle Configurator Developer

Throughout this section, references to the test configurator means test instances of a runtime Oracle Configurator launched from Oracle Configurator Developer by clicking the **Test** button.

15.3.1 [Merlin]

This section lists default DBOwner and JdbcUrl parameters for Oracle Configurator Developer. When a required parameter is missing in the [DSN], then the value found in [Merlin] is used.

DBOwner in [Merlin]

The parameter `DBOwner` specifies the default username for the owner of the Oracle Configurator schema that the `spx.ini` file accesses when starting up Oracle Configurator Developer. Users log into Oracle Configurator Developer with the schema owner name and the password.

This parameter must be updated to specify the actual DBOwner of the Oracle Configurator schema containing your Oracle Configurator Developer Model(s). Every database specified by a Data Source Name (see Section 15.3.2, "[DSN]", below) is associated with this DBOwner, unless another DBOwner is specified explicitly for both Oracle Configurator Developer and the test configurator with a specific DSN setting.

The DBOwner is to the Oracle Configurator Developer as FNDNAM is to the rest of Oracle Applications. However, to prevent limitations in Oracle Configurator Developer functionality, you should log in to Oracle Configurator Developer as an Oracle Applications FND user. To do this, the datasource description in the `spx.ini` file on the client machine where Oracle Configurator Developer is installed must provide additional gateway parameters (gwyuid and gwypass) that specify the Oracle public gateway login information for FND authentication. See DBOwner in [dsn] on page 15-8 for an example of the usage of the gwyuid and gwypass parameters.

JdbcUrl in [Merlin]

The **JdbcUrl** entry in this section indicates the default JDBC connection URL that is used when testing a DHTML user interface in Oracle Configurator Developer. If the database instance does not specify a JdbcUrl, then the JdbcUrl value in [Merlin] will be used.

15.3.2 [DSN]

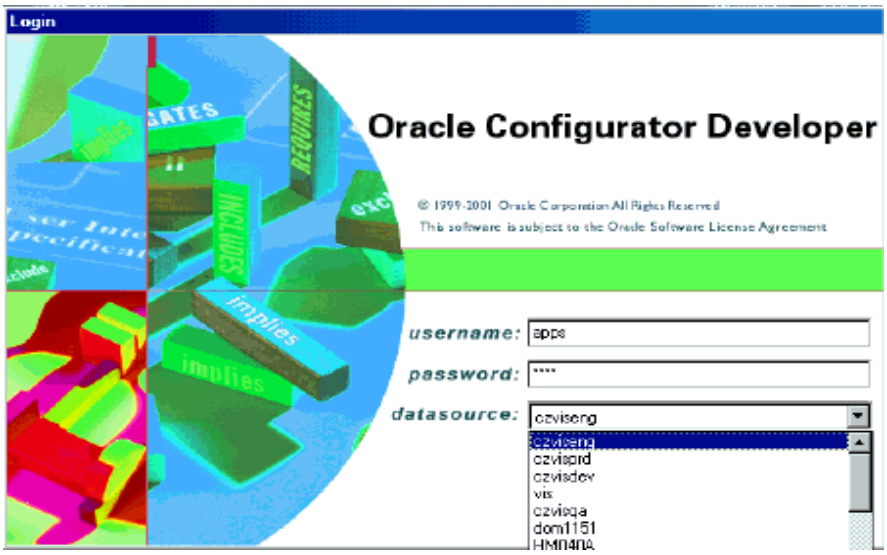
This section lists the Data Source Names (DSN) of the Oracle Configurator schemas available to Oracle Configurator Developer.

The DSN of the Oracle Configurator schema used with Oracle Configurator Developer must be listed here. The entries in the DSN section then appear in the Oracle Configurator Developer list of available data sources when you log in to Oracle Configurator Developer. A section must also be added for each DSN entry of the server DBOwner by which users will access the server Oracle Configurator schema (see Section 15.3.3, "[odbc-dsn]" on page 15-8).

Oracle Configurator Developer and the test Configurator require that the DSNs defined in the `spx.ini` file point to an installed Oracle Configurator schema. The DSNs set in the `spx.ini` file must also be registered in the ODBC Administrator for each machine running Oracle Configurator Developer and the test Configurator. See the *Oracle Configurator Installation Guide* for information about registering DSNs and database connectivity.

Figure 15–2, "OCD Logon Screen" shows the DSN entries found in a `spx.ini` file.

Figure 15–2 OCD Logon Screen



You must also edit the `spx.ini` file and add entries for each of the DSNs available to an instance of the test configurator and DBOwner. The DSNs listed in Example 15–2 on page 15-4 are examples of those installed as part of the Oracle Configurator installation. (See Section 15.3.2, "[DSN]" on page 15-7.)

Additional parameters may be defined specifically for manipulating the behavior of the test configurator instance.

15.3.3 [odbc-dsn]

This is a discrete [dsn] section to an Oracle Configurator schema instance. The settings in this section override the default settings in [Merlin]. This instance must also be specified in the [DSN] section.

DBOwner in [dsn]

In order for the developer to log into the Oracle Applications database, you must provide the owner of the schema as the value of the DBOwner parameter. Each discrete DSN section specifies the DBOwner by which the Oracle Configurator schema associated with the DSN will be accessed. This value must be the same as the value of the FNDNAM parameter in the Oracle Applications DBC file.

```
[dsn1]
DBOwner=DBOwner
JdbcUrl=Jdbc_connection
gwyuid=gateway_user_id
gwypass=gateway_password
applid=application_id
respid=responsibility_id
```

JdbcUrl in [dsn]

Each discrete `[odbc-dsn]` section indicates how the runtime Oracle Configurator should connect to the database. If you are using only Dynamic HTML in a browser for unit testing, you must add a JdbcUrl entry. If you are using a Java Applet window for unit testing, this parameter is not required. If you are using DHTML and this value is not specified in the discreet dsn section, then there is no entry in the Test dialog box in Oracle Configurator Developer, and the JdbcUrl value in [Merlin] is then used.

Use the format: `jdbc:oracle:thin:@dbhost:dbport:SID`, where dbhost is the name of the local machine, dbport is the port where your service is running, and SID is the service name of your database instance.

Gwyuid and Gwypass in [*odbc-dsn*]

Oracle Configurator uses Oracle Applications FND authentication for user authentication for Oracle Configurator Developer and the runtime Oracle Configurator. Therefore, you must add two entries to each discrete [*odbc-dsn*] section that provide Oracle gateway information, gwyuid and gwypass. **Gwyuid** is the public Oracle gateway username and **gwypass** is the public Oracle gateway password that grants access to the Oracle Applications log on form. Gwyuid and gwypass should be the same as the default username and password in your Oracle Applications environment file.

The mserv and training sections in Example 15–2 on page 15-4 include these entries for Oracle Applications user authentication.

APPLID and RESPID in [*DSN*]

When unit testing a Model from Oracle Configurator Developer using a DHTML window, or calling Oracle Configurator from another client (calling) application, you may want to retrieve translated messages (FND messages). In order to do this, an icx_session_ticket must be passed to the XML initialization message from Oracle Configurator Developer. An icx_session_ticket helps to maintain the connection parameters established during connection (user ID, responsibility ID, and application ID).

The APPLID (application ID) and RESPID (responsibility ID) parameters work in combination with the FND_USER_ID and are used to generate the icx_session_ ticket information for the database session initialization message. This combination determines the language in which FND messages are returned. The APPLID is the identifier for the client application and RESPID is the identifier for the associated responsibility. These parameters are integer values. You can find the appropriate integer value for the application and responsibility that applies to your calling application in the Oracle Applications FND_RESPONSIBILITY tables.

If these parameters are not specified, one cannot launch Configurator from the Test button as there is no ICX ticket in the init string and it will fail when logging in. Oracle Configurator Developer passes an initialization message using only the database connection (dbc) file. See the *Oracle Configurator Installation Guide* for information about the dbc file. See Chapter 9, "Session Initialization" for information about the initialization message.

InitServletURL

This parameter specifies which OC Servlet to use for testing the model with the **Test** button in Oracle Configurator Developer. When Launch=1, this parameter must be set to specify the URL of the servlet that is generating the DHTML in the browser. If

Launch=2 than this parameter must be set to specify the URL of the servlet that is generating the Java applet.

InitCodeBaseURL

This parameter must be set to specify the URL of the location of the class or Java archive (.jar) files for the Java Applet. For example:

```
InitCodeBaseUrl=http://www.myhost.com:port/applet
```

> **Note:** The `InitCodeBaseUrl` is used only when unit testing a Model and UI in a Java applet using the **Test** button in Configurator Developer.

Launch

This parameter determines the type of runtime environment to launch when using the Test button in OCD.

Launch=1 specifies Dynamic HTML. When using the Dynamic HTML InitServletURL must be set. See InitServletURL for details.

Launch=2 specifies the Java Applet. When using the Java Applet InitServletURL must be set. See InitServletURL for details. Additionally, when unit testing in Oracle Configurator Developer using the Java Applet, the **InitCodeBaseURL** parameter must be set. See InitCodeBaseURL for details.

You can also specify the test environment in the Options window in Oracle Configurator Developer. Changes you make to the **InitServletURL** or **InitCodeBaseURL** in the Options window are automatically updated in the `spx.ini` file. For any Oracle Configurator Developer session, the setting in the Options window is stored in memory and used when launching the test Configurator. If you manually edit the `spx.ini` file, the change does not take affect until you restart Configurator Developer.

For more information about the Options window, see the *Oracle Configurator Developer User's Guide*.

15.3.4 [Design Chart]

This section sets the alpha symbols used to indicate defining (DEF=) and secondary (SEC=) Optional Features in Oracle Configurator Developer Design Chart configuration rules. Example 15–2 shows, you can assign multiple symbols. If you use multiple defining and secondary Features in a Design Chart rule, the defining

(DEF=) set of symbols must be completely different than the secondary (SEC=) set of symbols.

15.3.5 Data Source-Specific Test Sessions

If you wish, you can set separate testing parameters for each data source used by Oracle Configurator Developer. These parameters are set in a [dsn} section of the `spx.ini` file. The values for the following parameters can be different for each data source:

Launch
InitServletURL
InitCodeBaseURL
jinitVersion
jinitClassID

Syntax:

```
[odbc_dsn]
DbOwner=db_owner
JdbcUrl=jdbc:oracle:thin:@db_host:db_port:sid
Launch=option
InitServletURL=http://web_
host:port/configurator/oracle.apps.cz.servlet.UiServlet
jinitVersion=version
jinitClassID = clsid_of_jinitiator
```

If you are using Microsoft Internet Explorer when unit testing a configuration model in Oracle Configurator Developer, you must specify the both the version and class ID of JInitiator in your `spx.ini` file, using the jinitVersion and jinitClassID parameters. For example:

```
[dsn1]
...
jinitVersion=1.1.8.7
jinitClassID = clsid:ff348b6e-fd21-11d4-a3f0-00c04fa32518
```

If you omit a value for `jinitClassID` from the `spx.ini` file, Oracle Configurator Developer assumes that you are using version 1.1.8.7 of JInitiator (required by Oracle Applications), and uses a hardcoded class ID which specifies this version.

Note: It is recommended the user specify the value found in the jinit-version.txt on their client machine.

You can determine the version number and class ID of your installed version of JInitiator by examining the file jinit-version.txt file in the doc subfolder of your JInitiator installation folder. For example:

```
C:\Program Files\Oracle\JInitiator 1.1.8.7\doc\jinit-version.txt
```

When you upgrade, you can use your existing spx.ini file. If you use the **Tools** > **Options** > **Test** dialog in Oracle Configurator Developer to change any test options, then your spx.ini file is rewritten to place the test parameters in the section for the data source that you logged in to.

Example

Here are some sections of spx.ini before you log in:

```
[sid01]
DbOwner=apps
JdbcUrl=jdbc:oracle:thin:@db01:1521:sid01

[sid02]
DbOwner=apps
JdbcUrl=jdbc:oracle:thin:@db04:1521:sid02

[sid03]
Launch=1
InitServletURL=http://webhost05:8010/configurator/oracle.apps.cz.servlet.UiServl
et
jinitVersion=1.1.8.7
jinitClassID = clsid:ff348b6e-fd21-11d4-a3f0-00c04fa32518
```

Here are the same sections of spx.ini after you log in to the data source sid01 and change the *web_host* specified in the InitServletURL parameter:

```
[sid01]
DbOwner=apps
JdbcUrl=jdbc:oracle:thin:@db01:1521:sid01
Launch=1
InitServletURL=http://webhost12:4199/configurator/oracle.apps.cz.servlet.UiServl
et
jinitVersion=1.1.8.7
jinitClassID = clsid:ff348b6e-fd21-11d4-a3f0-00c04fa32518

[sid02]
DbOwner=apps
JdbcUrl=jdbc:oracle:thin:@db04:1521:sid02
```

```
[sid03]
Launch=1
InitServletURL=http://webhost05:8010/configurator/oracle.apps.cz.servlet.UiServl
et
jinitVersion=1.1.8.7
jinitClassID = clsid:ff348b6e-fd21-11d4-a3f0-00c04fa32518
```

15.3.6 Parameterized Startup of Oracle Configurator Developer

You can start up Oracle Configurator Developer with predefined parameters. This enables you to provide preset values for the mandatory login parameters (user, password, data source name), and bypass the OCD login screen where you normally enter them. You can also control logging with a parameter. For more information, see the *Oracle Configurator Installation Guide*.

16

Customizing the HTML Template

This chapter describes some ways of customizing the HTML Template files that comprise part of the user interface for the runtime Oracle Configurator.

> **Note:** This document does not explain how to customize a user interface that is not generated through Configurator Developer, or that uses a non-Oracle host application.

- See Section 16.1, "How It Works" on page 16-1 for an explanation of the technology that underlies the tasks described in this chapter.
- See Section 16.2, "What You Do" on page 16-15 for a summary of the tasks related to this technology.
- See Section 16.3, "Customizing the HTML Template Files" on page 16-17 for details on customization tasks.
- See Section 16.4, "Using Meta-Events" on page 16-20 for description of a specialized topic.

16.1 How It Works

The HTML Template has a structure that supports the operation of the Oracle Configurator window.

Oracle Configurator includes a UI Server that renders a DHTML representation of an Oracle Configurator Model and user interface, as defined in Configurator Developer and published in the OC Servlet and User Interface.

Oracle Configurator provides a set of HTML files that make up the parts of a single HTML Template. This Template can be used as a model for producing alternate

Template designs. This enables you to incorporate the Oracle Configurator window into other host applications, retaining the functionality built into the Oracle Configurator window while adapting its look and feel to match the surrounding frames of the host application.

Custom Template versions can be created in any HTML editor or text editor.

16.1.1 Structure of the HTML Templates

It is necessary to understand the basic structure of the HTML Templates, in order to know which parts you can customize, and how.

The Templates are provided as a set of HTML files that define:

- A core set of required frames, which render the user interface that you generate through Oracle Configurator Developer. See Section 16.1.1.1 on page 16-2.
- A surrounding set of customizable frames, which fill out the Oracle Configurator window, and which contain action buttons, images, decorations, and the like. See Section 16.1.1.2 on page 16-5.

The rest of this section explains this structure.

16.1.1.1 Required Frames

Required template frames are those that are essential for supporting the operation of the DHTML components of the user interface of the runtime Oracle Configurator. The required frames must be incorporated into every customized Oracle Configurator window.

The required frames are grouped into a frameset called the **Inner Frameset**, which always consists of:

- The Content Frame, which is split into a Tree view and a Display view
- The Source, Proxy, and Heartbeat Frames

Figure 16–1 on page 16-3 shows the structure of the Inner Frameset. Table 16–1 on page 16-3 explains the elements of the Inner Frameset that are shown in Figure 16–1.

Figure 16–1 Structure of the Inner Frameset

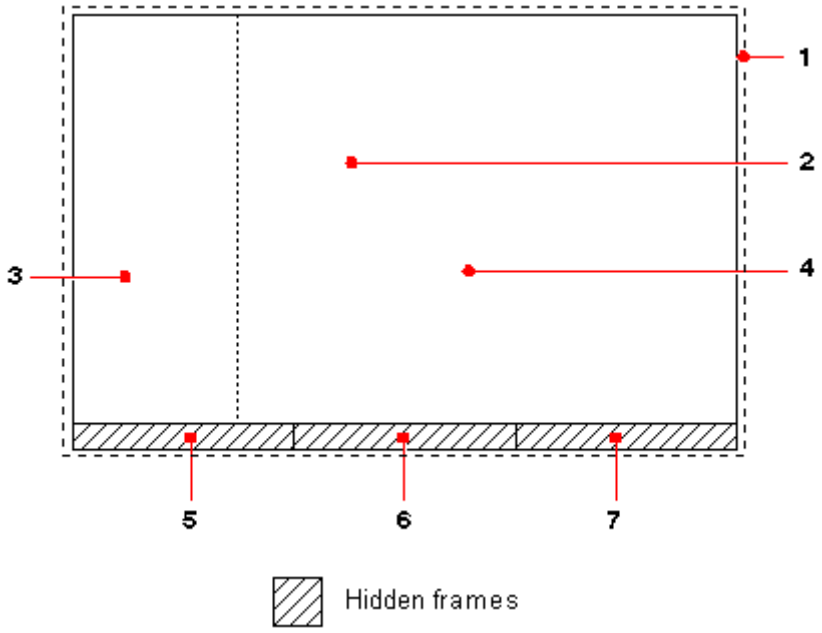


Table 16–1 Elements of the Inner Frameset

No.	Name	Definition
1	Inner Frameset	Contains the frames required by the runtime Oracle Configurator.
2	Content Frame	Contains the Tree View and the Display View, and coordinates the interaction between them. When the end user invokes a function to display the Summary screen, the Summary occupies this entire frame.
3	Tree View	Shows a representation of the Model tree. Each component screen in the user interface of the runtime Oracle Configurator is represented by an icon. The end user uses this tree to navigate through the runtime configuration model. This view can be hidden, as described in Section 16.2.2.1 on page 16-17.
4	Display View	Contains the Oracle Configurator user interface that you generate through Oracle Configurator Developer. You can only customize this user interface through Oracle Configurator Developer.

Table 16–1 (Cont.) Elements of the Inner Frameset

No.	Name	Definition
5	Source Frame	Contains the definitions of the JavaScript controls that provide the behavior of the DHTML user interface. These controls are defined in the file `czAll.js`, which is imported by this frame.
6	Proxy Frame	Functions as the communications center between the Oracle Configurator window and the OC Servlet (which communicates with the UI Server). Sends events from the client browser to the OC Servlet, and returns responses from the OC Servlet that update the browser window.
7	Heartbeat Frame	The Heartbeat Frame is used to communicate with the UI Server when the DHTML interface is used for guided selling in an Oracle Applications context. (For details on the heartbeat mechanism, see the *Oracle Configurator Installation Guide*.)

Hidden Frames

As indicated in Figure 16–1 on page 16-3, the Source, Proxy, and Heartbeat Frames are hidden from the end user. This is done by setting their height allocations to 0 (zero) in the Inner Frameset, which contains them.

Required Controls

The Inner Frameset does not contain the controls needed for working with the current configuration as a whole (for instance to save it at the end of a configuration session). Such controls are provided in the HTML Templates; whether they are placed in the Header Frame or the Footer Frame depends on the look and feel being deployed. See Section 16.1.2, "Look and Feel" on page 16-9 and Section 16.1.3, "The Action Buttons" on page 16-14.

If you are implementing a custom template that replaces the Header Frame or the Footer Frame, then you must provide the controls for triggering functions in the Template or in the host application. Triggering functions are:

- Submit (save the configuration and exit)
- Cancel (exit without saving)
- Show configuration or summary information (in alternation)
- Display ATP (Available To Promise) information, if it is provided by the host application

16.1.1.2 Customizable Frames

Customizable template frames are those that you can customize and augment. They are not required for the operation of the runtime Oracle Configurator, but you must supply some part of the functionality they provide in order to create a practical Oracle Configurator window.

The customizable frames are grouped into a frameset called the **Outer Frameset**, which consists of some combination of:

- The Header Frame
- The Footer Frame
- The Left and Right Frames

The customizable frames can be combined in different ways, to provide various graphical looks and feels. The Oracle Configurator HTML Templates provide several predefined combinations, which are described in Section 16.1.2, "Look and Feel" on page 16-9.

16.1.1.3 Guide to Modifying the Template Files

Table 16–2 lists the constraints on customizing the Template, and the files in which the elements of the Template are implemented. The hierarchical relationship of the framesets and frames is indicated in the layout of the table.

Table 16–2 Constraints on the HTML Template

No.	Req?	Cust?	Frameset/Frame				File
	No	Yes	Outer Frameset				`czBlafTemplate.jsp` (W) `czFormTemplate.jsp` (F) `czIFrame.htm`[1] (I)
1	Yes	No		Inner Frameset			`czFraTemplate.jsp`
2	Yes	No			Content Frame		`czCntnt.jsp`
3	Yes	No				Tree View	`cztree.jsp` (W) `czFormTree.jsp` (F) `czblank.jsp` (F) `czseperator.jsp` (F)

Table 16–2 (Cont.) Constraints on the HTML Template

No.	Req?	Cust?	Frameset/Frame				File
4	Yes	No				Display View	`czdisp.jsp`
5	Yes	Yes[2]			Hidden Frameset	Source Frame	`czSource.jsp`
6	Yes	No				Proxy Frame	`Proxy.class` (OC Servlet)
7	Yes	No				Heartbeat Frame	`czHeartBeat.jsp`
8	No	Yes		Header Frame			`czHeader.jsp` (W) `czFormHeader.jsp` (F) `czButtonBar.jsp` (I)
9	No	Yes[3]		Left Frame			`czLeft.jsp` (F)
10	No	Yes[4]		Right Frame			`czRight.jsp` (F)
11	No	Yes		Footer Frame			`czFormFooter.jsp` (F)

[1] This file is named `czIFrame.htm` for compatibility with certain host applications.

[2] See Section 16.3.1 on page 16-18 for the customizations allowed for this file.

[3] Do not customize this file if you are using the default Oracle Forms Look. See Section 16.1.2.3 on page 16-12.

[4] Do not customize this file if you are using the default Oracle Forms Look. See Section 16.1.2.3 on page 16-12.

Legend for Table 16–2

Column	Description
No.	The identification number in Figure 16–1 through Figure 16–6.
Req?	Is this frameset or frame required for a valid Template?
Cust?	Can this frameset or frame be customized?
Frameset / Frame	The name of the frameset or frame.The hierarchical relationship of the framesets and frames is indicated in the layout of the table. Containing elements are to the left of contained elements.

Column	Description
File	The name of the file that implements the frameset or frame. All Template files are located in the OA_HTML directory of the OC Servlet. See the *Oracle Configurator Installation Guide* for details. Some files are used only for a particular look and feel. These are indicated by a letter following the file name: ■ (W) = Web look and feel ■ (F) = Forms look and feel ■ (I) = IFrame look and feel See Section 16.1.2, "Look and Feel" on page 16-9.

Note: If you have recently upgraded your Oracle Configurator installation, and previously customized the HTML Template files, then your customizations are contained in HTML files, rather than in JSP files. In addition, some of the HTML files are stored in the US subdirectory of the OA_HTML directory. See Table 16–3 on page 16-7 for the correspondence between HTML and JSP files.

To continue to use your previously customized HTML files, you must set the property of the OC Servlet named cz.uiservlet.ignore_url_properties. See the *Oracle Configurator Installation Guide* for details.

Table 16–3 Correspondence of HTML to JSP Files

HTML Files	Java Server Page (JSP) Files
OA_HTML/czblank.htm	OA_HTML/czblank.jsp
OA_HTML/czCntnt.htm	OA_HTML/czCntnt.jsp
OA_HTML/czdisp.htm	OA_HTML/czdisp.jsp
OA_HTML/czFormTree.htm	OA_HTML/czFormTree.jsp
OA_HTML/czHeartBeat.htm	OA_HTML/czHeartBeat.jsp
OA_HTML/czseperator.htm	OA_HTML/czseperator.jsp
OA_HTML/czSource.htm	OA_HTML/czSource.jsp
OA_HTML/cztree.htm	OA_HTML/cztree.jsp

Table 16–3 (Cont.) Correspondence of HTML to JSP Files

HTML Files	Java Server Page (JSP) Files
OA_HTML/US/czBlafTemplate.htm	OA_HTML/czBlafTemplate.jsp
OA_HTML/US/czFormTemplate.htm	OA_HTML/czFormTemplate.jsp
OA_HTML/US/czFraTemplate.htm	OA_HTML/czFraTemplate.jsp
OA_HTML/US/czLeft.htm	OA_HTML/czLeft.jsp
OA_HTML/US/czRight.htm	OA_HTML/czRight.jsp

16.1.1.4 The Summary Screen

The Summary screen is implemented using the file `OA_HTML/czSummary.jsp`. However, it is not considered part of the customizable Template, and you cannot customize it. Its contents are controlled by the UI Server, which uses a variety of criteria, such as the value of CZ_PS_NODES.QUOTEABLE_FLAG, to determine which items to show on the Summary screen. See Section 5.3.2, "Identifying Data for a Custom Data Import" on page 5-19 for more details.

16.1.1.5 Interacting with the Oracle Configurator Servlet

You can determine which Template files are used for the Oracle Configurator window by indicating your choice to the OC Servlet.

When the OC Servlet starts a configuration session, it chooses the Template to use for the session in the following way:

1. If the initialization message contains the initialization parameter template_url, then that specified Template is used. See Chapter 9, "Session Initialization" for details about template_url.

   Using this parameter is the only way to specify the Template file `czIFrame.htm`, to obtain the IFrame look and feel. This method is used by the Oracle *i*Store application. This method is also illustrated in Section 16.4.3, "Meta-Event Example" on page 16-22.

2. If the template_url parameter is not provided, then the OC Servlet automatically chooses the Template file that corresponds to the look and feel that you selected when you generated the user interface with Oracle Configurator Developer:

   - For a user interface generated with the Oracle Web Look (also called browser look and feel (BLAF)): `czBlafTemplate.jsp`.

- For a user interface generated with the Oracle Forms Look: `czFormTemplate.jsp`.

3. The location of each of the Template files is specified by one of the following properties of the OC Servlet:

   - `cz.uiservlet.blaftemplateurl` for `czBlafTemplate.jsp`
   - `cz.uiservlet.formtemplateurl` for `czFormTemplate.jsp`
   - `cz.uiservlet.templateurl` for other Template files

   See the *Oracle Configurator Installation Guide* for information on how to set the values for these properties. If these properties are omitted, then the OC Servlet uses the appropriate Template file in the default location, which is the `OA_HTML` directory.

   You can use these properties to point to alternate versions of the Template files in alternate locations, for instance to provide Multiple Language Support.

When you choose a new Template file to be used for new sessions, or modify the definition of the current file, you must clear the cache of your browser in order to see the changes.

16.1.2 Look and Feel

The user interface of the runtime Oracle Configurator is designed to work with several predefined Templates, each with a particular look and feel.

- See Table 16–2 on page 16-5 for information on the files that implement each look and feel.
- See Section 16.1.1.5 on page 16-8 for information on how to choose which look and feel is used by the OC Servlet for a configuration session.
- See the *Oracle Configurator Developer User's Guide* for information on how to generate a user interface, and choose a look and feel for it.

16.1.2.1 Oracle Web Look

The Oracle Web Look is also called the browser look and feel (BLAF). This user interface designed to resemble Oracle's Web-based applications, and to appear in a secondary window of its own that is opened when the host application starts the runtime Oracle Configurator.

The Header Frame is the only customizable frame in this look and feel template. This Template places its action buttons in the Header Frame.

See Figure 16–2 on page 16-10 for an illustration of the structure of the Web Look template. See Figure 16–3 on page 16-11 for an example of how the Web Look template appears when it displays the runtime Oracle Configurator. The identification numbers in Figure 16–2 and Figure 16–3 correspond to the **No.** column in Table 16–2 on page 16-5.

Figure 16–2 Structure of the Web Look Template

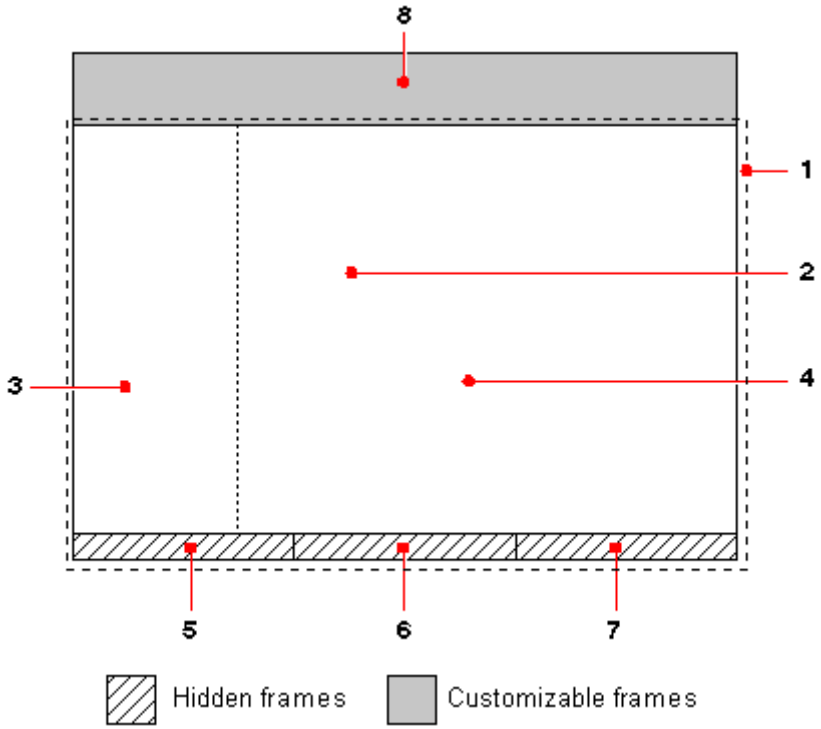
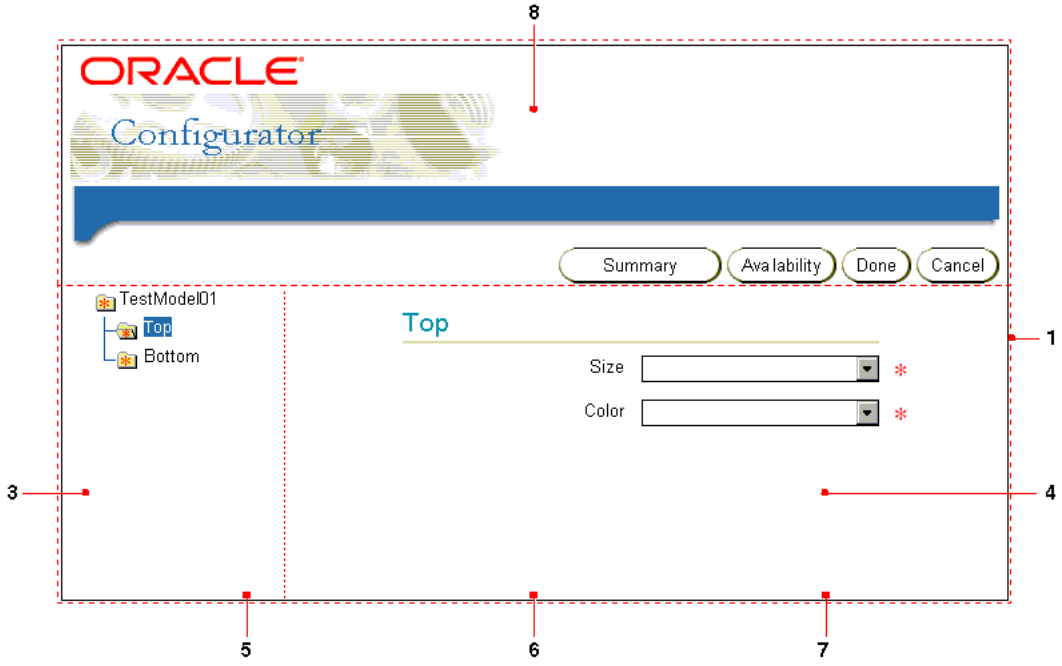


Figure 16–3 Example of Default Web Look Template



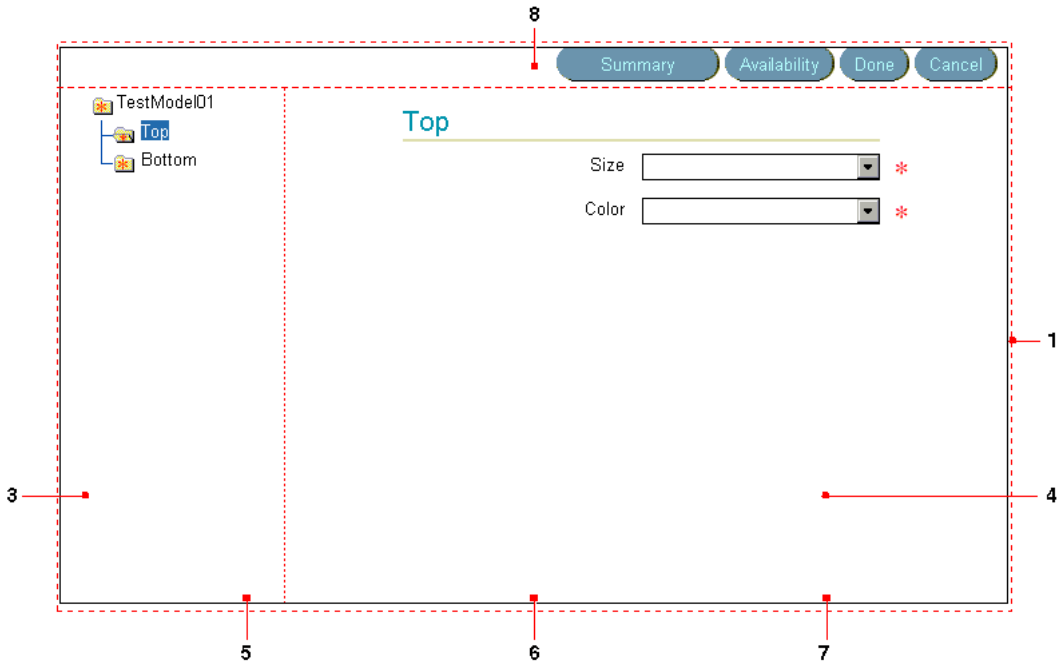
16.1.2.2 Oracle IFrame Look

The Oracle IFrame Look is similar to the Oracle Web Look, and uses the same basic structure (see Figure 16–2 on page 16-10). This look and feel is designed to appear in a frame that is internal to the window used by the host application, so it minimizes the size and decoration of the Header Frame. Note that the name "IFrame" denotes "internal frame", and is not related to the IFRAME element of HTML.

Like the Oracle Web Look, this user interface places its action buttons in the Header Frame, but provides a distinctive appearance for them.

See Figure 16–4 on page 16-12 for an example of how the IFrame Look template appears when it displays the runtime Oracle Configurator. The identification numbers in Figure 16–2 and Figure 16–4 correspond to the **No.** column in Table 16–2 on page 16-5. Note that the Header Frame lacks a screen title and decoration in the Header Frame; contrast this to the Web Look in Figure 16–3 on page 16-11.

Figure 16–4 Example of Default IFrame Template



16.1.2.3 Oracle Forms Look

The Oracle Forms Look is designed to resemble the user interface seen in Forms-based Oracle Applications such as Order Management, and to appear in a secondary window of its own that is opened when the host application starts the runtime Oracle Configurator.

This look and feel template includes customizable Header and Footer Frames. Do not customize the Left and Right Frames; doing so interferes with the alignment of the elements on the screen. This Template places its action buttons in the Footer Frame.

See Figure 16–5 on page 16-13 for an illustration of the structure of the Forms Look template. See Figure 16–6 on page 16-14 for an example of how the Web Look template appears when it displays the runtime Oracle Configurator. The identification numbers in Figure 16–5 and Figure 16–6 correspond to the **No.** column in Table 16–2 on page 16-5.

Figure 16–5 Structure of the Forms Look Template

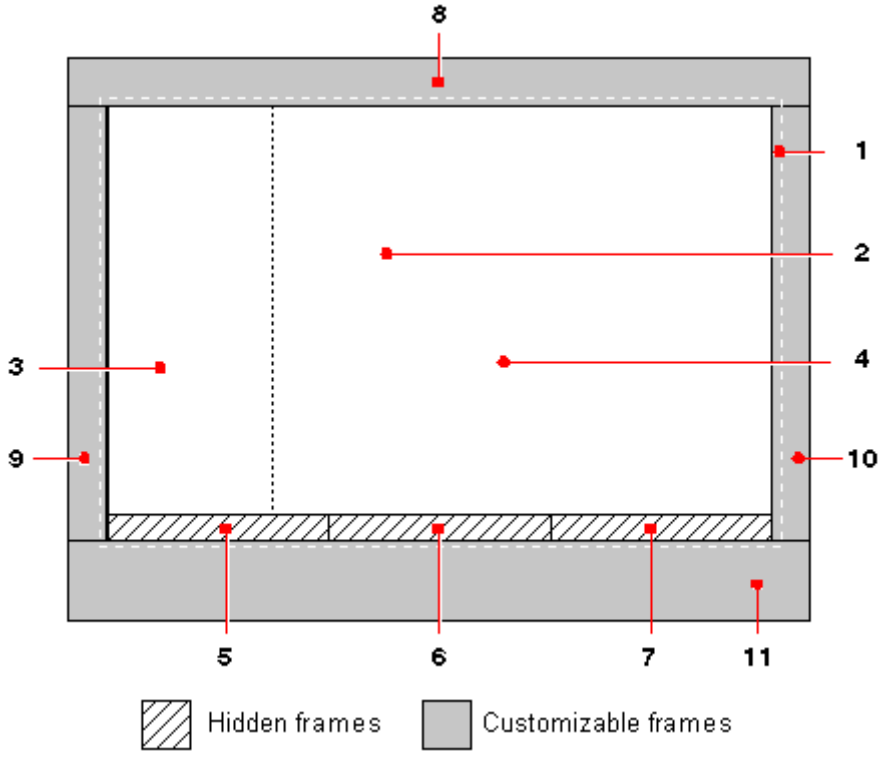
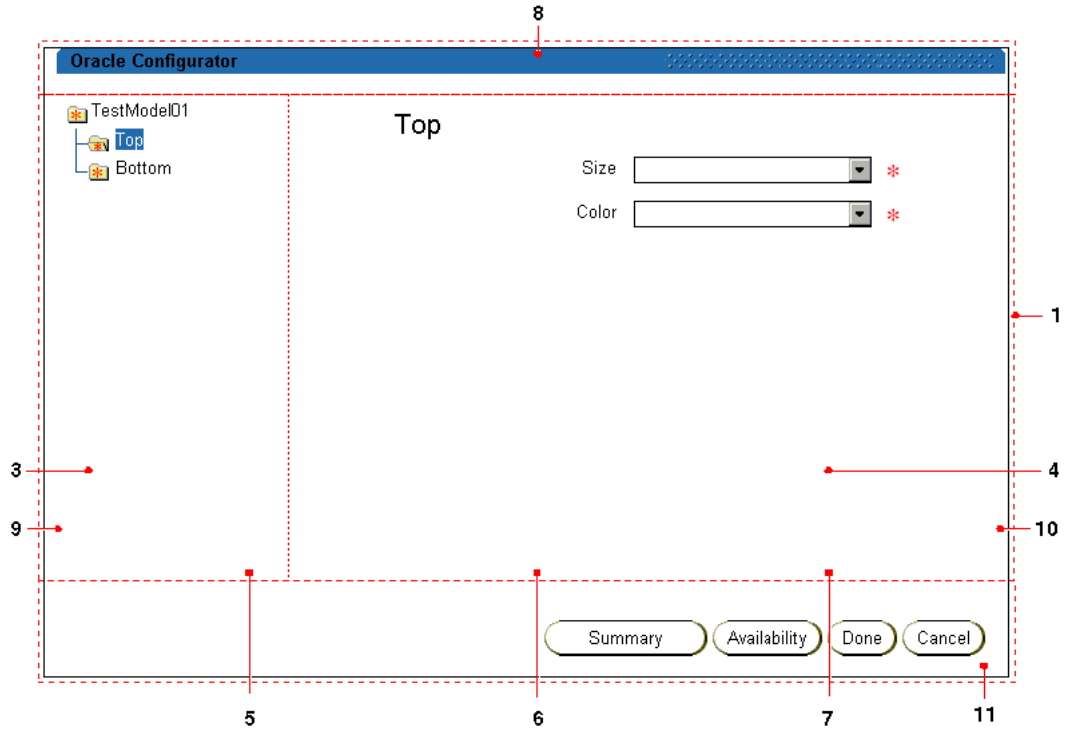


Figure 16–6 Example of Default Forms Look Template



16.1.3 The Action Buttons

Each of the HTML Templates provides a frame which it implements the graphical display of action buttons and the JavaScript definitions that trigger the actions themselves. The requirement for these buttons is noted in Required Controls on page 16-4.

The location and style of the action buttons depends on the look and feel of the Template being employed. See Section 16.1.2 on page 16-9 for information about look and feel.

You can customize the text of the buttons, as described in Section 16.3.2.3 on page 16-20.

The actions provided by the action buttons are described in Table 16–4 on page 16-15.

Table 16–4 Action Buttons in the Button Frame

Button Name	Description
Summary / Configuration	This button is scripted to toggle the Content Frame between the User Interface generated through Configurator Developer and the Summary screen. The script changes the caption on the button to read "Summary" when the Oracle Configurator UI is displayed and "Configuration" when the Summary screen is displayed.
Availability	This button is scripted to populate the current display with ATP (Available To Promise) dates. When triggered for the Summary screen, it refreshes the entire Summary page with data that includes the ATP information, if it is provided.
Done	This button is scripted to save the configuration, terminate the session, and return control to the host application.
Cancel	This button is scripted to terminate the session and return control to the host application without saving the configuration.

If you need to provide your own actions and buttons you can add them in the appropriate button frame. You may be able to provide the functionality you need by using meta-events, which are covered in Section 16.4 on page 16-20.

Warning: If you modify the existing button functions, the Oracle Configurator window may not operate correctly or at all.

The User Interface generated through Configurator Developer is able to include buttons for navigation (such as Next, Previous, and Home), so buttons are not provided in the Template for those actions. See the *Oracle Configurator Developer User's Guide* for more information.

16.2 What You Do

This section summarizes the tasks and directs you to sources of more information.

16.2.1 Customization Tasks

This section guides you toward the best way to customize the appearance of the Oracle Configurator window.

- You can customize the decorations in the Header and Footer Frames. You can also customize the decorations in the Left and Right Frames if you are not using the Oracle Forms Look.
- You can customize the buttons in the button frame (Header Frame or Footer Frame, depending on the look and feel).
- You can add new buttons in the button frame.
- You *cannot* modify the Content Frame, since its contents are automatically generated by the OC Servlet, based on a definition of the User Interface that you generate with Oracle Configurator Developer.
- You can customize a Template use meta-events, as described in Section 16.4, "Using Meta-Events" on page 16-20.

See Section 16.3, "Customizing the HTML Template Files" on page 16-17 for specific instructions, and Section 16.1, "How It Works" on page 16-1 for the necessary background.

16.2.2 Customizing the Default User Interface in Oracle Configurator Developer

You do not have to define a user interface for the Oracle Configurator window; a default one is already available if you used Oracle Configurator Developer (OCD) to generate a UI. The Test button in OCD produces an Oracle Configurator window that duplicates the one that a host application would produce, if it were not customized outside the Content Frame. Remember that you can only customize the elements inside the Display View of the Content Frame by using OCD. For details, see the sections of the *Oracle Configurator Developer User's Guide* that cover the user interface.

If you want to call Functional Companions from your user interface, define controls to call them in Configurator Developer. See the *Oracle Configurator Developer User's Guide* and the *Oracle Configuration Interface Object (CIO) Developer's Guide* for details.

There are a number of customizations to the default User Interface that rely on the HTML Template but are accomplished by means other than directly modifying the Template files:

- Hiding the Model Tree
- Controlling Expansion of New Component Instances

16.2.2.1 Hiding the Model Tree

You can hide the Tree view of the configuration model that is displayed in the Content Frame. To do so:

- In Oracle Configurator Developer, switch to the User Interface module, then adjust the control named **Allocate % of Display Width to Navigation Frame**. See the *Oracle Configurator Developer User's Guide* for details.

16.2.2.2 Controlling Expansion of New Component Instances

You can control whether new component instances added to the configuration are expanded in the Navigation Tree when an end user clicks an **Add** button at runtime. To do so:

- Set the Oracle Configurator Servlet property named `cz.uiserver.add_instance_expansion_policy`. See the *Oracle Configurator Installation Guide* for details on this property.

16.2.3 Restrictions on the Oracle Configurator Window

You cannot use BMP (Windows bitmap) files in your user interface for the Oracle Configurator window, since this file format is not compatible with rendering in a DHTML context. The Oracle Configurator window can use GIF, JPG, and other formats compatible with Web browsers.

16.3 Customizing the HTML Template Files

See Section 16.1.1.3, "Guide to Modifying the Template Files" on page 16-5 for a list of the Template files that can be modified. The template that you are most likely to modify is the Header Frame.

> **Warning: If you modify any of the template files, be sure to make backups. The template files are overwritten whenever you reinstall Oracle Configurator.**

The customizable template files are commented for your guidance.

If you are deploying in multiple languages, then the customizable HTML template files installed in the `OA_HTML/US` directory that contain text visible to the end user must be translated and installed in the appropriate directory, such as `OA_HTML/F` (French) or `OA_HTML/JA` (Japanese).

16.3.1 Specifying the Location of Media Files

When you design a User Interface in Configurator Developer, you can specify the location of custom image media files for your host application: icons, backgrounds, button shapes, and the like. When your host application uses the DHTML Oracle Configurator window, the UI Server strips the path information (which is stored in the database) from the name of these media files, and uses only the file name itself.

The HTML Template files set a global variable `IMAGESPATH`. `IMAGESPATH` points to the location of the media files and is read by the JavaScript controls that populate the Oracle Configurator window. By default, `IMAGESPATH` is set to `/OA_MEDIA/`, which is located under the document root directory for the OC Servlet. (Unless the Web server is configured to search elsewhere, it serves HTML documents from the document root directory.) See the *Oracle Configurator Installation Guide* for more details about `OA_MEDIA`.

If you have custom media files to use, you can store them in `OA_MEDIA`. If you wish to use a different location, it is recommended that you add a directory under `OA_MEDIA`, then modify the HTML Template to reflect that location. For example, to move your media files to a directory `alt` under the standard directory `OA_MEDIA`, do the following:

1. Copy your custom media files and the OC media files to the new directory.
2. Change this code in `czSource.jsp`, to reflect the location of the media files:

```
//Modify it depending on your server installation directory.
//** Do not forget to put the trailing SLASH at the end
var IMAGESPATH = '/OA_MEDIA/alt/';
var SOURCEPATH = '/OA_HTML/';
```

3. Restart the OC Servlet.

16.3.2 Customizing Text in the Oracle Configurator Window

You can customize some of the text elements that are displayed in the Header and Footer Frames in the Web Look and Forms Look templates (described in Section 16.1.2.1 on page 16-9 and Section 16.1.2.3 on page 16-12). You may need to do this for internationalization purposes, such as MLS and NLS. See Chapter 14, "Multiple Language Support" on page 14-1 for details on MLS (Multiple Language Support).

You can customize the following text elements in the Oracle Configurator window:

- The title displayed at the top of the window. See Section 16.3.2.1 on page 16-19.

- The status messages that display while the window is working. See Section 16.3.2.2 on page 16-19.
- The labels of the action buttons. See Section 16.3.2.3 on page 16-20.

These customizable text elements are stored in the table FND_NEW_MESSAGES in the Oracle Applications database. The column MESSAGE_NAME contains the key that identifies the text. The column MESSAGE_TEXT contains the customizable text itself. You can modify the text of titles, messages, or button labels by using an UPDATE statement that changes the value of MESSAGE_TEXT, for the specified MESSAGE_NAME.

16.3.2.1 Customizing the Screen Titles

The screen title is displayed in the Header Frame in the Web Look and Forms Look templates (described in Section 16.1.2.1 on page 16-9 and Section 16.1.2.3 on page 16-12). You can also customize the title of the Summary screen (although you cannot customize the other parts of the Summary).

The default screen titles are described in Table 16–5 on page 16-19, along with the database keys required to update them.

Table 16–5 Screen Titles for Header Frame

Template	Default Title	Database Key
Forms Look	Oracle Configurator	CZ_GENERIC_ORACLE_SP_TITLE
Web Look	Configurator	CZ_PRODUCT_TITLE
Summary	Configuration Summary	CZ_SUMMARY_TITLE

You can view the default title text in SQL*Plus with a query like the following:

```
SELECT MESSAGE_NAME,SUBSTR(MESSAGE_TEXT,1,60) FROM FND_NEW_MESSAGES WHERE
MESSAGE_NAME LIKE 'CZ_%_TITLE';
```

16.3.2.2 Customizing the Status Messages

You can customize the text of the status messages that are displayed in the Header Frame when the runtime Oracle Configurator is loading or processing. The status messages are described in Table 16–6 on page 16-20.

The default status messages are described in Table 16–6 on page 16-20, along with the database keys required to update them.

Table 16–6 Status Messages for Header Frame

Displayed When...	Default Message	Database Key
Loading	Loading Configurator.....	CZ_LOADING_STATUS_MSG
Processing	processing.....	CZ_PROCESS_STATUS_MSG

You can view the default message text in SQL*Plus with the following query:

```
SELECT MESSAGE_NAME,SUBSTR(MESSAGE_TEXT,1,30) FROM FND_NEW_MESSAGES WHERE
MESSAGE_NAME LIKE 'CZ_%_STATUS_MSG';
```

16.3.2.3 Customizing the Button Labels

You can customize the text of the labels of the action buttons that are displayed in the Header Frame or Footer Frame. The buttons are rendered in the Oracle Configurator window by combining GIF images and text within a table cell. The images are stored in the `OA_MEDIA` directory.

The default button labels are described in Table 16–7 on page 16-20, along with the database key required to update them.

Table 16–7 Button Labels for Header Frame

Default Label	Database Key
Configuration	CZ_CONFIGURATION_HEADER_BUTTON
Summary	CZ_SUMMARY_HEADER_BUTTON
Availability	CZ_AVAILABILITY_HEADER_BUTTON
Done	CZ_DONE_HEADER_BUTTON
Cancel	CZ_CANCEL_HEADER_BUTTON

You can view the default label text in SQL*Plus with the following query:

```
SELECT MESSAGE_NAME,SUBSTR(MESSAGE_TEXT,1,30) FROM FND_NEW_MESSAGES WHERE
MESSAGE_NAME LIKE 'CZ_%_BUTTON';
```

16.4 Using Meta-Events

You can customize the HTML Templates to invoke meta-events to perform certain actions during a configuration session.

16.4.1 About Meta-Events

A meta-event is an event that is sent programmatically from a host application through the runtime Oracle Configurator to the UI Server. Meta-events are used to execute defined Oracle Configurator behavior, or to provide a specialized action that is not provided by the uncustomized Oracle Configurator window.

Meta-events are expressed as XML elements. They are passed as arguments to the JavaScript method `UiBroker.postClientMessage()`. The `UiBroker` object is instantiated by the Source Frame of the HTML Template.

The meta-events supported for your use are listed in Table 16–8 on page 16-21.

Table 16–8 Supported Meta-Events

Meta-Event and Syntax	Description
`<navigate name='screen'/>`	Navigates to the named screen in the UI tree. Screen names are defined in the User Interface module of Oracle Configurator Developer (OCD). If *screen* is a Component that can have multiple instances, then you must navigate to them with `<next/>` and `<previous/>`.
`<next/>`	Navigates to the next screen in the UI tree (depth first[1]). At the last screen, navigates to the root screen.
`<previous/>`	Navigates to the previous screen in the UI tree (depth first). At the root screen, navigates to the last screen.
`<refresh-frames/>`	Refreshes the current screen.
`<submit/>`	Saves the configuration and exits the session.
`<save exit='state' type='type'/>`	Saves the current configuration, optionally exiting the screen. Provides options for how to save. The values for the `exit` parameter are `true` and `false`. Saving with `exit` set to `true` is equivalent to the `<submit/>` meta-event. The values for the save `type` parameter are described under save_config_behavior on page 9-29.
`<cancel/>`	Cancels the current session, without saving the configuration.

Table 16–8 (Cont.) Supported Meta-Events

Meta-Event and Syntax	Description
`<execute-fc name='fc_name'/>`	Executes the named event that is associated with a Functional Companion. The value for the `name` parameter is the name used to register the Functional Companion. See step 4. under Section 16.4.4 on page 16-25. The Type of the Functional Companion should be Event-Driven.

[1] "Depth first" means that each child node is visited before any siblings.

16.4.2 Invoking Meta-Events

In order to invoke a meta-event from a frame in the Template outside the Content Frame, use JavaScript syntax like that shown in Example 16–1.

Example 16–1 Invoking a Meta-Event with JavaScript

```
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<eventName parameter=\'value\'/>')" >Link Text</a>
```

Meta-events are invoked by passing an XML message to the `UiBroker` object that is instantiated by the Source Frame of the HTML Template. Therefore, the JavaScript call must traverse the structure of the Template frames from the invoking frame to the Source Frame, as shown in Example 16–1 on page 16-22.

You can use any valid means of invoking the JavaScript call. For simplicity, Example 16–1 shows the use of the HTML <A> tag with a text link.

Note the use of escaped quotation marks around the string values of meta-event parameters in Example 16–1.

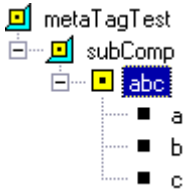
For a full example of invoking typical meta-events, see Section 16.4.3 on page 16-22.

16.4.3 Meta-Event Example

This section demonstrates a typical way of using the meta-events described in Table 16–8 on page 16-21.

1. Define a Model with the structure shown in Figure 16–7. The Feature named `abc` is a List of Options with Minimum = 1 and Maximum = 1. Generate the Active Model and Generate a User Interface for the Model.

Figure 16–7 Model Structure for Meta-Event Example



2. Create a custom frame file named `metaTest.html`, and place it in the `OA_HTML` directory of your Oracle Configurator Servlet. Enter the HTML code shown in Example 16–2 into `metaTest.html`. See Example 16–1, "Invoking a Meta-Event with JavaScript" on page 16-22 for information on the syntax.

Example 16–2 Custom Frame for Meta-Events (metaTest.html)

```
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<!-- links for invoking meta-events -->
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<navigate name=\'metaTagTest\'/>')" >Navigate To Root</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<navigate name=\'subComp\'/>')" >Navigate To subComp</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<execute-fc name=\'metaTagTest\'/>')" >Turn Off Selected Option</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<next/>')" >Next</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<previous/>')" >Previous</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<refresh-frames/>')" >Refresh</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<cancel/>')" >Cancel</a>
```

```
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<submit/>')" >Submit</a>
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMessage
('<save exit=\'false\' type=\'new_config\'/>')" >Save without Exit</a>

</body>
</html>
```

3. Substitute your custom frame file (`metaTest.html`) for the Header Frame in the IFrame look and feel template.

   To make this substitution, edit the file `OA_HTML/czIFrame.htm`, and substitute `metaTest.html` for `czButtonBar.jsp`, as shown in the following HTML code fragment.

   ```
   <FRAME name="header" id="header" SRC="/OA_HTML/metaTest.html"
   marginwidth="10" marginheight="0" frameborder="0" framespacing="0"
   border="0" scrolling="no" align="left" valign="top">
   ```

4. Modify the initialization message in a test page for your Oracle Configurator Servlet to add the template_url parameter. Specify the location of the `OA_HTML/czIFrame.htm` file, as shown in the following code fragment, with your site-specific values for *`host`* and *`port`*.

   ```
   <param name="template_url">http://host:port/configurator/OA_
   HTML/czIFrame.htm</param>
   ```

   See Example 9–3 on page 9-6 for a typical test page for the OC Servlet. You can also use the Test button in Oracle Configurator Developer and then copy and modify the test page that it produces in `C:\temp\dhtmlhtm.htm`.

5. Restart your OC Servlet, then use the test page to start the runtime Oracle Configurator. The Oracle Configurator window should display the IFrame look and feel, with your custom frame file in the Header Frame, in place of the original file (`czButtonBar.jsp`).

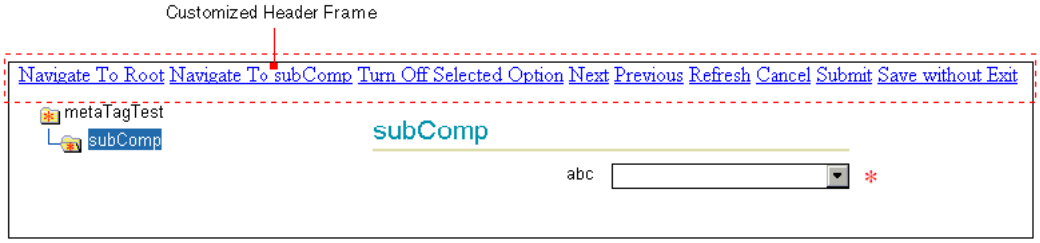
6. In the Oracle Configurator window, select an Option for the Feature `abc`, and try clicking the links in the customized Header Frame. Compare the effects to Table 16–8 on page 16-21.

Table 16–9 Link Actions for Meta-Event Example

Link Text	Expected Effect
Navigate To Root	Navigate to the screen `metaTagTest`.
Navigate To subComp	Navigate to the screen `subComp`.
Turn Off Selected Option	Invokes a Functional Companion that deselects the Option you have chosen for the Feature `abc`. You must first create the Functional Companion according to Section 16.4.4 on page 16-25.
Next	Navigates to the next screen.
Previous	Navigates to the previous screen.
Refresh	Refreshes the screen.
Cancel	Cancels the current session, without saving the configuration.
Submit	Saves the configuration and exits the session.
Save without Exit	Saves the a new revision of the configuration but does not exit the session.

7. After you click the link labeled `Navigate To subComp`, the Oracle Configurator window should look like Figure 16–8 on page 16-25.

Figure 16–8 Testing Meta-Events in the Oracle Configurator Window



16.4.4 Invoking a Functional Companion with a Meta-Event

Among the meta-events that you can send to Oracle Configurator is:

```
<execute-fc name='fc_name'/>
```

which executes the Functional Companion named by *fc_name*.

The following steps describe how to create the Functional Companion that performs the task described in Table 16–9 on page 16-25 (deselecting the current Option).

1. Compile the source code shown in Example 16–3 on page 16-26 as `changeAbc.java`. See the *Oracle Configuration Interface Object (CIO) Developer's Guide* for details.
2. Install the class file in the class path of the Oracle Configurator Servlet. See the *Oracle Configurator Installation Guide* for details.
3. In Oracle Configurator Developer, add a Functional Companion rule to the Model shown in Figure 16–7 on page 16-23. The name of the rule is not important for this example.

   For the Base Component, choose the Model (`metaTagTest`). For the Program Name, enter `changeAbc`. For the Type, choose Event-Driven.
4. The name specified in the JavaScript invocation of the `execute-fc` meta-event must match the event ID string specified as the first argument to the `addUserInterfaceEventListener()` method in the Functional Companion. This is shown in the following code fragments.

   The JavaScript invocation is shown in this fragment from Example 16–2 on page 16-23:

```
...
<a href="javascript:void(0);"
onClick="javascript:parent.frames.template.czSrc.getUiBroker().postClientMes
sage('<execute-fc name=\'metaTagTest\'/>')" >Turn Off Selected Option</a>
...
```

   The Functional Companion method is shown in this fragment from Example 16–3 on page 16-26:

```
...
      mUi.addUserInterfaceEventListener("metaTagTest", this);
...
```

   Note that the name used to register the Functional Companion for the <execute-fc> meta-event (in this case, `metaTagTest`) is *not* necessarily the name of the Java class that implements the Functional Companion (in this case, `changeAbc`).

Example 16–3 Functional Companion for a Meta-Event (changeAbc.java)

```
import java.io.PrintWriter;
import java.io.*;
```

```
import javax.servlet.http.*;
import oracle.apps.cz.cio.FunctionalCompanion;
import oracle.apps.cz.cio.AutoFunctionalCompanion;
import oracle.apps.cz.cio.IUserInterface;
import oracle.apps.cz.cio.*;
import oracle.apps.cz.cio.IRuntimeNode;
import com.sun.java.util.collections.List;
import com.sun.java.util.collections.Iterator;
import oracle.apps.cz.utilities.CheckedToUncheckedException ;
import oracle.apps.cz.utilities.Assert;
import java.awt.Toolkit;

public class changeAbc extends AutoFunctionalCompanion implements
IUserInterfaceEventListener {
  IUserInterface mUi;
  IOptionFeature mChangeAbc;

  public void initialize(IRuntimeNode node, String name, String description, int
id) {
    super.initialize(node, name, description, id);
    mUi = this.getRuntimeNode().getConfiguration().getUserInterface();
    if(mUi != null) {
      mUi.addUserInterfaceEventListener("metaTagTest", this);
    }

    IRuntimeNode r = this.getRuntimeNode();
    IRuntimeNode c = (IRuntimeNode)r.getChildren().get(0);
    try {
      mChangeAbc = (IOptionFeature)c.getChildByName("abc");
            }
      catch(Exception e) {
        throw new RuntimeException(e.getMessage() );
      }
  }
  public void handleUserInterfaceEvent(IUserInterfaceEvent event) {
      try {
        if (event.getName().equals("metaTagTest")) {
          IOption o = mChangeAbc.getSelectedOption();
          if (o != null) o.deselect();
        }
      }
      catch(Exception e) {
        throw new RuntimeException(e.getMessage() );
      }
  }
```

```
}
```

17

Publishing Configuration Models

This chapter assumes that you are familiar with the publishing process, which is explained in detail in the *Oracle Configurator Developer User's Guide.*

This chapter includes information about:

- Planning publications
- The database tables used during the publishing process
- How hosting applications select a publication at runtime
- Maintaining publication

17.1 Planning Publications

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 calling applications.

To use publications effectively, it is necessary to plan and design how each publication will be used. How you define publications depends on whether you are working with imported BOM Models or configuration models that are created from scratch in Oracle Configurator Developer.

As part of your planning, you should identify what you need to accomplish and how the Oracle Configurator publication functionality can help you achieve the results you desire. Once you have determined how the publication functionality applies to your situation, identify the necessary tasks in Oracle Applications and Oracle Configurator Developer.

To prevent end users from receiving errors, you should plan for and try to create publications for all circumstances in which hosting applications will access your configuration models. Applications that can host a runtime Oracle Configurator can

access only a single publication of a configuration model. If several publications exist for one Model, only one of those publications can be valid at a time. 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. (The terms "source publication" and "remote publication" are defined in Section 17.2 on page 17-5.)

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 on multiple development instances from publishing Models to the same target instance. You can only be sure that you are not creating publications with overlapping applicability parameters in the same database if you copy all publications from one development instance. For this reason, Oracle strongly recommends that you publish from only *one* source database.

The `PublishingCopyRules` setting in CZ_DB_SETTINGS determines whether the Model's rules are copied along with the Model's structure and User Interface definition during publication. For more information, see Section 4.4.3.20 on page 4-15.

Your project design should account for how you use hosting applications, Usages, effective date ranges, Effectivity Sets, 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 configuration models and test them in one database, but run your production environment on a separate, production database?
- What is your publication plan, relative to the databases you are using?
- Are you going to use Usages to determine which users can access a publication?
- Are you going to use effectivity dates, Effectivity Sets, and Usages within configuration models to limit the availability of Model structure or configuration rules?
- What hosting applications will access your publications?

(For a list of hosting applications that support Oracle Configurator, see the *Oracle Configurator Release Notes*.)

- Is your hosting application registered in Oracle Applications?

  All applications that are part of Oracle Applications are registered when you install Oracle Applications. If you are using a custom Web application, you must register it in Oracle Applications. See the *Oracle Applications System Administrator's Guide* for more information.

- How will you use publication mode to restrict access to testers and end users? For instance, when testing on the production database before going live, publication mode set to Test excludes end users from accessing those Models, even though they exist in the production database.

17.1.1 How Hosting Applications Select a Publication

All applications that can host an Oracle Configurator window 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, the corresponding configuration model and UI appears in the Configurator window. If no matching publication is found, Oracle Configurator displays an error.

For example, in your business you know that two different hosting 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 **Application** applicability parameter to ONT for the first publication, and IBE for the second (these are the application short names for Order Management and *i*Store, respectively). An Oracle Applications user whose Responsibility is assigned to Order Management selects Model M1 in the Sales Orders window, and clicks the **Configure** button.

Using the information in the initialization message, the OC Servlet selects the only publication in the database that:

- Has the Application parameter set to OM
- Matches all of the other parameters specified in the session initialization message

Then, the OC Servlet displays the configuration model and UI that you defined specifically for orders placed from Order Management in the Configurator window. To view the UI created for *i*Store users, the initialization message must contain

For detailed information about the initialization message, see Chapter 9, "Session Initialization".

For information about entering applicability parameters when creating a publication, see the *Oracle Configurator Developer User's Guide*. See the *Oracle Configurator Release Notes* for a listing of short names of Oracle Applications that support Oracle Configurator.

17.1.1.1 Example: Using Usages to Select a Publication

Your company makes and sells cars. The Environmental Protection Agency requires that cars sold in California meet more rigorous emissions guidelines than other states in the U.S. Therefore, you must install a different type of exhaust system on any car sold in California. When defining the configuration models that represent each car that you sell, you create a different Component for the two types of exhaust systems. You then create two Usages, CAL-EPA for cars sold in California and US-EPA for cars sold in other states, and assign them to your exhaust system Components.

When an end user wants to configure a car, the hosting application first collects information about the user, including (in this example) the state in which the car will be sold. The host application uses this information to construct the intialization message. If the car is sold in California, then the CAL-EPA Usage is set in the initialization message. Otherwise, the US-EPA Usage is set in the initialization message. The host application then passes the initialization message to the Configurator window, which then selects and displays the Model with the appropriate exhaust system.

17.1.1.2 Publication Profile Options

If a Usage or publication mode is not specified in the session initialization message, and the host application is registered within Oracle Applications, the following profile options provide default values for these parameters:

- CZ:Publication Usage
- CZ:Publication Lookup Mode

For more information about these profile options, see the *Oracle Configurator Installation Guide*.

17.1.2 Publishing and Model References

If you are publishing a configuration model that has References to other Models, 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 will not be able to configure the non-published Model in a runtime Oracle Configurator.

The availability of referenced Models is controlled by the Usages applicability parameter. See the *Oracle Configurator Developer User's Guide* for more information on the Usages applicability parameter.

17.2 Defining a Publication

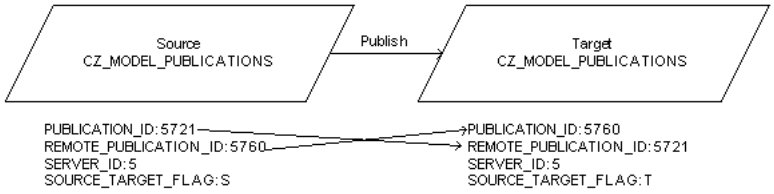
Defining a publication in Oracle Configurator Developer creates a **source publication** with a unique publication ID in the CZ_MODEL_PUBLICATIONS table in the development database instance. When the publication and Model data is exported to the target database instance (by running a publication concurrent program in Oracle Applications), a record of the publication is created on the target database; this is called a **remote publication** (see Section B.3 on page B-9). Each value in a source publication record corresponds to a value in the remote publication record.

For example, when a configuration model is published, the following data is created:

- 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

Figure 17–1 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.

Figure 17–1 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 and examples of overlapping publications, see the *Oracle Configurator Developer User's Guide*.

17.2.1 Tables Used in Publishing

Following are the database tables that are used during the publishing process:

- CZ_MODEL_PUBLICATIONS
- CZ_MODEL_USAGES
- CZ_PB_CLIENT_APPS
- CZ_PB_MODEL_EXPORTS
- CZ_PUBLICATION_USAGES
- CZ_MODEL_APPLICABILITIES_V

For detailed information about the publishing tables (or any other tables in the CZ schema), go to Metalink, Oracle's technical support Web site, at:

`http://www.oracle.com/support/metalink/`

17.2.2 Publication Attributes

A publication is defined by its attributes and applicability parameters. When you create a new publication or edit an existing publication, these attributes are found on the **General** tab in the Edit Publication window in Configurator Developer. A publication's attributes define the runtime circumstances and environment in which the published configuration model is accessible.

A publication's attributes include:

- Model
- User Interface
- Database Instance
- Product ID

Defining publications is described in the *Oracle Configurator Developer User's Guide*.

17.2.2.1 Model

The **ID** column in the Model Publishing window corresponds to the MODEL_ID field in the CZ_MODEL_PUBLICATIONS table. This MODEL_ID is the CZ_DEVL_PROJECTS.DEVL_PROJECT_ID that returns CZ_DEVL_PROJECTS.NAME. This is the Model name that appears in the Configurator Developer Repository window.

17.2.2.2 User Interface

If the configuration model specified by the publication has multiple User Interfaces, the list of available User Interfaces in the Model Publishing window comes from the CZ_UI_DEFS table. The available User Interfaces are determined by the selected configuration model.

17.2.2.3 Database Instance

In the Model Publishing window, the list of values for this parameter includes all databases listed in the CZ_SERVERS table. This parameter indicates the database in which the publication and model data will exist.

If configuration model is published to a remote database instance, then the remote database instance must be defined and enabled. For more information about defining and enabling a remote server, see Section B.2, "Server Administration Concurrent Programs" on page B-4.

17.2.2.4 Product ID

Product ID is a designation relevant only when publishing in Oracle Configurator Developer. There is no corresponding Product node in a configuration model's structure.

The **Product ID** field in the Model Publishing window displays different information depending on whether the Model specified is an imported BOM Model or a Configurator Developer (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 in CZ_MODEL_PUBLICATIONS.PRODUCT_KEY. 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, then the **Product ID** field is the name of the root Model node that was entered in Configurator Developer. In this case, you can modify the **Product ID**.

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, 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. Refer to the *Oracle Configurator Implementation Guide* for more information about the **Product ID** field.

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 Chapter 9, "Session Initialization".

17.2.3 Publication Applicability Parameters

Applicability parameters determine the availability of a publication to hosting applications. Applicability parameters include:

- Mode
- Valid From / Valid To
- Usages
- Applications
- Languages

This section contains information about how the publication applicability parameters are used internally by the runtime Oracle Configurator. 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 hosting application interacts with these parameters to select a publication, see Section 9.5.2 on page 9-9.

17.2.3.1 Mode

Values for this parameter include Test or Production. For information about the publication_mode parameter in the session initialization message, see Section 9.6 on page 9-16. See the *Oracle Configurator Installation Guide* for information of the Oracle Applications profile option CZ: Publication Lookup Mode.

17.2.3.2 Valid From/Valid To

A publication's effective dates are stored in the columns APPLICABLE_FROM and APPLICABLE_TO in the CZ_MODEL_PUBLICATIONS table.

17.2.3.3 Usages

The Usages defined in Configurator Developer are stored in CZ_MODEL_USAGES, and are displayed in the list of values when assigning Usages to a publication in the Model Publishing window. The Usages assigned to a publication are stored in CZ_PUBLICATION_USAGES.

If the hosting application does not provide a Usage in its initialization message, the Oracle Configurator Servlet uses the value of the Oracle Applications profile option CZ: Publication Usage. For more information about this profile option, see the *Oracle Configurator Installation Guide*.

For an example of how Usages are used by a hosting application at runtime, see Section 17.1.1.1 on page 17-4.

For general information about Usages and how to define them in Configurator Developer, see the *Oracle Configurator Developer User's Guide*.

17.2.3.4 Applications

The FND_APPLICATION table contains the application ID of each application that is registered in Oracle Applications. This numeric value corresponds to the application's short name, which appears in the list of values for the **Application** applicability parameter in the Configurator Developer Model Publishing window. (A system administrator defines an application's short name when registering it in Oracle Applications.)

For a complete list of the applications that support Oracle Configurator, as well as each application's short name and application ID, see the *Oracle Configurator Release Notes*.

When you save a publication, the specified applications are stored in the CZ_PB_CLIENT_APPS table.

Warning: If you want to add applications to FND_APPLICATION, be aware that APPLICATION_ID is a sequence ID and if you are exporting publications from one database to another, the publication must be able to access precisely the same applications defined in FND_APPLICATION in both databases.

17.2.3.5 Languages

The Languages applicability parameter determines in which language(s) the publication is available. This value is stored in the LANGUAGES column in CZ_MODEL_APPLICABILITIES_V.

For information about Multiple Language Support (MLS), see Chapter 14.

17.3 Publishing a Configuration Model

After a source publication has been defined in Oracle Configurator Developer, the Model data must be copied to the target database by running one of the publication concurrent programs. This creates the remote publication on the target database. When the concurrent program completes successfully, the remote publication can then be accessed by hosting applications such as Oracle Order Management and *i*Store.

For more information, see Section B.3, "Configuration Model Publication Concurrent Programs" on page B-9.

17.3.1 Checking BOM and Model Similarity

The publication concurrent programs call the Model synchronization concurrent programs as part of the publishing process. If there are key discrepancies between the BOM Model and the configuration model to be published, then an error message is logged by the publication concurrent program and the configuration model is not published.

Example 17–1 illustrates an error found in CZ_DB_LOGS file when attempting to publish a configuration model (publication ID = 28261).

Example 17–1 Publishing Error when Checking BOM 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 information, see Section 7.2.1 on page 7-2.

17.4 Maintaining Publications

In a typical organization, 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.

17.4.1 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, in the Model Publishing window in Configurator Developer.

Table 17–1 lists each status and the corresponding tasks that you can perform.

Table 17–1 Publication Status and Valid Operations

Status	New or New Copy	Edit	Republish	Delete
Complete	Y	Y	Y	Y
Pending	Y	Y	N	Y
Update Pending	Y	N	N	N
Processing	Y	N	N	N
Error	Y	N	N	Y

Configurator Developer updates the status of all publications whenever you open the Model Publishing window or click **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 the Model Publishing window. When the Oracle Applications concurrent program successfully copies the Model data to the publication target database, the publication status changes to Complete.
- **Pending Update**: A request to update the existing publication has been created in the Model Publishing window. When the Oracle Applications concurrent program successfully copies the Model data to the publication target database, the publication's status changes to Complete.
- **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 publication in the Model Publishing window but another Configurator Developer user updates the Model before the Oracle Applications concurrent program is complete. To create the publication, select the record, then click either **New Publication** or **New Copy** from the Model Publishing window.

Refer to the *Oracle Configurator Developer User's Guide* for information about creating and editing publications.

17.4.2 Editing Publications

Editing a publication refers to modifying its applicability parameters, which changes the publication's availability to hosting applications. 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). Therefore, it is *not* necessary to run one of the publication concurrent programs.

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.*

17.4.3 Disabling, Deleting, and Re-enabling Publications

You can make a publication unavailable to hosting applications by disabling it in the Oracle Configurator Developer Model Publishing window. When a publication is disabled, it still appears in the Model Publishing window and its status does not change. 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 Oracle Configurator Developer Model Publishing window, and it cannot be recovered.

See the *Oracle Configurator Developer User's Guide* for more information.

17.4.4 Republishing

You must republish a configuration model when changes are made to the Model structure, rules, or User Interface in Configurator Developer. 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 Update Pending in the Publishing window, and has a STATUS of 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 a previously published configuration model is modified in Configurator Developer and is then republished, 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.

Note: If you set the profile option CZ: Populate Decimal Quantity Flags to `Yes` and then import or refresh your BOM Models, you must republish Model publications to ensure that they use the new setting. Decimal quantities are explained in Section 5.2.7.6 on page 5-11.

17.4.5 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 publishing window in a simple SQL*Plus query returns the UI_DEF_ID. 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.

Example 17–2 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 Section 18.3, "Reference for the CZ_CF_API Package" on page 18-8.)

17.4.6 Managing 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 **Valid From** and **Valid To** applicability parameters were not specified.

To retrieve orders for the previously published Model, you must:

1. Edit the first published Model's **Valid To** applicability parameter to have an end date.
2. Republish the Model.
3. Publish the newer Model with the **Valid From** applicability equal to the **Valid To** end date of the first published Model.

See the *Oracle Configurator Developer User's Guide* to learn how to perform these tasks.

17.4.7 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

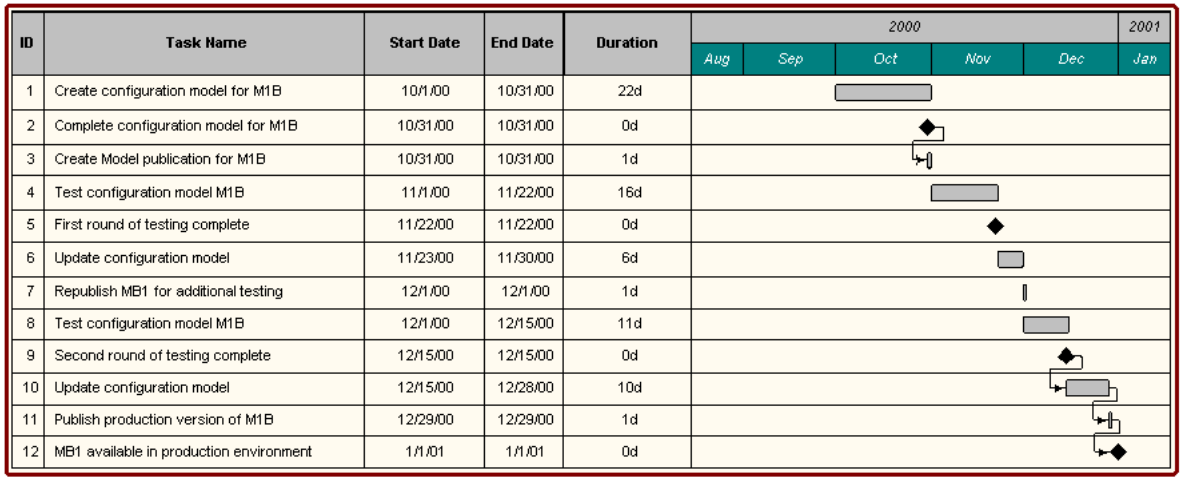
For more information, see Chapter 7, "Synchronizing Data".

17.4.8 Example of Maintaining Publications

This section provides an example of how an organization might 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 organization's production environment and there can be no period of time when either one or the other Model is unavailable to customers.

Figure 17–2 provides an overview of how this organization plans to develop, test, and release M1B into production.

Figure 17–2 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	12/28/00	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 Figure 17–2.

1. Using Configurator Developer, the development team creates a new configuration model to reflect the new product's features and enhancements. The Model is unit tested periodically within Oracle Configurator Developer (using the Test module), 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 with a publication mode of Test. The publication is effective immediately and no end date is required since it can be modified at any time. The Applications and Usage parameters specify which hosting applications and 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 hosting application that the testers use selects the configuration 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 updates to the User Interface.
7. When changes to the Model are complete, developers republish the Model. This copies any new or modified data to the specified database so 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 hosting 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 organization, 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:
    a. Change the Mode for publication P1 from Test to Production
    b. Change the **end date** of the now obsolete publication for Model M1A to 12:00:00 a.m. on 01/01/01
    c. Specify a **start date** for publication P1 of 12:00:00 a.m. on 01/01/01

Notes

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.

If the development and testing environments are on different databases, developers must create a new publication with a publication Mode of Production (all of the other applicability parameters would be the same as publication P1). Whenever you need to copy Model data and specify a different User Interface, database instance, or applicability parameters, you must create a new publication.

See the *Oracle Configurator Developer User's Guide* for more information.

18

Programmatic Tools for Development

This chapter describes programmatic tools that you can use primarily to develop a configuration model and deploy a runtime Oracle Configurator.

For information on tools for maintaining a deployed runtime Oracle Configurator, see Chapter 19, "Programmatic Tools for Maintenance".

18.1 Overview of the CZ_CF_API Package

The programmatic tools that you use while developing or deploying a runtime Oracle Configurator are provided in the PL/SQL package CZ_CF_API.

18.1.1 Purpose of the Package

The CZ_CF_API package contains a set of APIs that enable you to various perform tasks such as the following:

- Copying and deleting configurations
- Determining default dates used by the runtime Oracle Configurator
- Establishing session identity
- Identifying publications
- Validating configurations

18.1.2 Overview of Procedures and Functions

Table 18–1 on page 18-2 summarizes and categorizes the procedures and functions available in the package CZ_CF_API.

These procedures and functions are described in individual detail in Section 18.3, "Reference for the CZ_CF_API Package" on page 18-11.

Table 18–1 Overview of Procedures and Functions in the Package CZ_CF_API

Category	API Name	P/F[1]
Working with Common Bills See Section 18.2.5.	COMMON_BILL_FOR_ITEM	P
Copying and Deleting Configurations See Section 18.2.4.	COPY_CONFIGURATION	P
	COPY_CONFIGURATION_AUTO	P
	DELETE_CONFIGURATION	P
Setting Configuration Dates See Section 18.2.2.	DEFAULT_NEW_CFG_DATES	P
	DEFAULT_RESTORED_CFG_DATES	P
Establishing Session Identity See Section 18.2.1.	ICX_SESSION_TICKET	F
Identifying Publications See Section 18.2.6.	CONFIG_MODEL_FOR_ITEM	F

Table 18–1 (Cont.) Overview of Procedures and Functions in the Package CZ_CF_API

Category	API Name	P/F[1]
	CONFIG_MODEL_FOR_PRODUCT	F
	CONFIG_MODELS_FOR_ITEMS	F
	CONFIG_MODELS_FOR_PRODUCTS	F
	CONFIG_UI_FOR_ITEM	F
	CONFIG_UI_FOR_ITEM_LF	F
	CONFIG_UI_FOR_PRODUCT	F
	CONFIG_UIS_FOR_ITEMS	F
	CONFIG_UIS_FOR_PRODUCTS	F
	MODEL_FOR_ITEM	F
	MODEL_FOR_PUBLICATION_ID	F
	PUBLICATION_FOR_ITEM	F
	PUBLICATION_FOR_PRODUCT	F
	PUBLICATION_FOR_SAVED_CONFIG	F
	UI_FOR_ITEM	F
	UI_FOR_PUBLICATION_ID	F
Validating Configurations See Section 18.2.3.	VALIDATE	P

[1] P = procedure, F = function

18.1.3 Installation of the Package

This package is installed in the Oracle Applications database as part of Oracle Configurator.

- If you installed a new instance of Oracle Applications, then this package was installed by using Oracle Rapid Install.
- If you installed Oracle Configurator in an existing instance of Oracle Applications, then this package was installed by applying the appropriate Oracle Configurator patch.

See the *Oracle Configurator Installation Guide* for details about installing Oracle Configurator.

18.1.4 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 Table 18–2, " References for Working with PL/SQL Procedures and Functions", which suggests relevant topics in the Oracle Documentation Library.

Table 18–2 References for Working with PL/SQL Procedures and Functions

See this topic ...	In this reference document ...
User-defined data types	*Oracle8i Concepts*
Procedures and packages	
Using procedures and packages	*Oracle8i Application Developer's Guide - Fundamentals*
Calling stored procedures	
Understanding the Oracle programmatic environments	
Language elements	*PL/SQL User's Guide and Reference*
Packages	
Index-by tables	
Collections and records	
User-defined subtypes	
Using SQL*Plus	*SQL*Plus User's Guide and Reference*
UTL_HTTP	*Oracle8i Supplied PL/SQL Packages Reference*

18.2 Choosing the Right Tool for the Job

These procedures and functions are described in detail in Section 18.3.2, "Procedures and Functions in the CZ_CF_API Package" on page 18-9.

18.2.1 Establishing Session Identity

Use the following function to establish the identity of a Oracle Applications database session:

- ICX_SESSION_TICKET

18.2.2 Setting Configuration Dates

Use these procedures to determine the dates that would be used for configurations:

- DEFAULT_NEW_CFG_DATES
- DEFAULT_RESTORED_CFG_DATES

18.2.3 Validating Configurations

Use this procedure to validate a configuration:

- VALIDATE

18.2.4 Copying and Deleting Configurations

Use these procedures to copy and delete configurations:

- COPY_CONFIGURATION
- COPY_CONFIGURATION_AUTO
- DELETE_CONFIGURATION

18.2.5 Working with Common Bills

Use this procedure to retrieve a common bill:

- COMMON_BILL_FOR_ITEM

18.2.6 Identifying Publications

After publishing Models, you can verify whether a publication lookup will succeed for a given set of applicability parameters. See Section 18.2.6.2 on page 18-6 for details about specifying applicability parameters.

18.2.6.1 Functions for Identifying Publications

Use these functions to look up publications for a given set of applicability parameters:

- CONFIG_MODEL_FOR_ITEM
- CONFIG_MODEL_FOR_PRODUCT

- CONFIG_MODELS_FOR_ITEMS
- 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
- MODEL_FOR_ITEM
- MODEL_FOR_PUBLICATION_ID
- PUBLICATION_FOR_ITEM
- PUBLICATION_FOR_PRODUCT
- PUBLICATION_FOR_SAVED_CONFIG
- UI_FOR_ITEM
- UI_FOR_PUBLICATION_ID

18.2.6.2 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 Chapter 9, "Session Initialization" for more information.

Table 18–3 on page 18-7 lists the applicability parameters that many of the functions and procedures in this package use to search for Models, UIs, and publications.

Table 18–3 Applicability Parameters for Publication Searches

Parameter in this package	Data type	Parameter in OCD[1]	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 (Valid From, Valid To)	Provide a date that falls inside the applicable range for the publication. Use the standard Oracle TO_DATE function to format the date.
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'`

Table 18–3 (Cont.) Applicability Parameters for Publication Searches

Parameter in this package	Data type	Parameter in OCD[1]	Description
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'`

[1] These names are for fields in the Model Publishing window of Oracle Configurator Developer.

18.2.6.3 List Parameters

In order to reduce the number of function calls when an application needs to 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, also known as index-by tables:

- date_tbl_type
- number_tbl_type
- varchar2_tbl_type

Parameters in this package that are of one of these list types do not default to NULL.

See Section 18.3.1, "Custom Data Types" on page 18-9 for the definition of these types.

18.3 Reference for the CZ_CF_API Package

- This section provides descriptions of each of the procedures and functions in the CZ_CF_API package. These procedures and functions are listed alphabetically in Table 18–5, " Procedures and Functions in the Package CZ_CF_API" on page 18-10
- Descriptions of the custom data types defined in the package are provided in Section 18.3.1, "Custom Data Types" on page 18-9.
- For a basic example of how to call one of the functions in the CZ_CF_API package, see Example 18–1, "Using the UI_FOR_PUBLICATION_ID Function" on page 18-59.

- See also Section 18.1, "Overview of the CZ_CF_API Package" on page 18-1.

18.3.1 Custom Data Types

Table 18–4, " Custom Data Types in the Package CZ_CF_API" 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 Section 18.1.4, "References for Working with PL/SQL Procedures and Functions" on page 18-4:

Table 18–4 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 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)

18.3.2 Procedures and Functions in the CZ_CF_API Package

This section provides descriptions of each of the procedures and functions in the CZ_CF_API package, arranged alphabetically. These procedures and functions are listed alphabetically in Table 18–5, " Procedures and Functions in the Package CZ_CF_API".

Table 18–5 Procedures and Functions in the Package CZ_CF_API

API Name	P/F[1]
COMMON_BILL_FOR_ITEM on page 18-11	P
CONFIG_MODEL_FOR_ITEM on page 18-13	F
CONFIG_MODEL_FOR_PRODUCT on page 18-17	F
CONFIG_MODELS_FOR_ITEMS on page 18-15	F
CONFIG_MODELS_FOR_PRODUCTS on page 18-19	F
CONFIG_UI_FOR_ITEM on page 18-21	F
CONFIG_UI_FOR_ITEM_LF on page 18-24	F
CONFIG_UI_FOR_PRODUCT on page 18-27	F
CONFIG_UIS_FOR_ITEMS on page 18-29	F
CONFIG_UIS_FOR_PRODUCTS on page 18-31	F
COPY_CONFIGURATION on page 18-33	P
COPY_CONFIGURATION_AUTO on page 18-36	P
DEFAULT_NEW_CFG_DATES on page 18-39	P
DEFAULT_RESTORED_CFG_DATES on page 18-41	P
DELETE_CONFIGURATION on page 18-43	P
ICX_SESSION_TICKET on page 18-45	F
MODEL_FOR_ITEM on page 18-47	F
MODEL_FOR_PUBLICATION_ID on page 18-49	F
PUBLICATION_FOR_ITEM on page 18-50	F
PUBLICATION_FOR_PRODUCT on page 18-52	F
PUBLICATION_FOR_SAVED_CONFIG on page 18-54	F
UI_FOR_ITEM on page 18-56	F
UI_FOR_PUBLICATION_ID on page 18-58	F
VALIDATE on page 18-61	P

[1] P = procedure, F = function

COMMON_BILL_FOR_ITEM

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 function to retrieve the common bill's details if the Model has not been published.

Considerations Before Running

Prerequisites
None.

Timing
Intended as a utility method during development.

Dependencies
None.

Restrictions and Limitations
None.

Warnings
None.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE common_bill_for_item ( in_inventory_item_id IN NUMBER,
                                 in_organization_id IN NUMBER,
                                 common_inventory_item_id OUT NUMBER,
                                 common_organization_id OUT NUMBER);
```

Table 18–6 on page 18-12 describes the parameters for the COMMON_BILL_FOR_ITEM procedure.

Table 18–6 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

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:

```
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;
```

Table 18–7 on page 18-14 describes the parameters for the CONFIG_MODEL_FOR_ITEM function.

Table 18–7 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

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

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:

```
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;
```

Table 18–8 on page 18-16 describes the parameters for the CONFIG_MODELS_FOR_ITEMS function.

Table 18–8 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number_tbl_type	in	List of registered IDs of applications for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2_tbl_type	in	List of Usage names to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2_tbl_type	in	List of publication modes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2_tbl_type	in	List of language codes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

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

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:

```
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;
```

Table 18–9 on page 18-18 describes the parameters for the CONFIG_MODEL_FOR_PRODUCT function.

Table 18–9 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

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

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 + and "P" (Production) respectively.

Syntax and Parameters

The syntax for this function is:

```
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;
```

Table 18–10, " Parameters for the CONFIG_MODELS_FOR_PRODUCTS Function" on page 18-20 describes the parameters for the CONFIG_MODELS_FOR_PRODUCTS function.

Table 18–10 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
config_lookup_date	date_tbl_type	in	List of dates to search for inside the applicable range for the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number_tbl_type	in	List of registered IDs of applications for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2_tbl_type	in	List of Usage names to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2_tbl_type	in	List of publication modes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2_tbl_type	in	List of language codes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

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

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:

```
FUNCTION config_ui_for_item (inventory_item_id IN NUMBER,
                             organization_id IN NUMBER,
                             config_lookup_date IN DATE,
                             ui_type IN OUT VARCHAR2,
                             calling_application_id IN NUMBER,
                             usage_name IN VARCHAR2,
                             publication_mode IN VARCHAR2 DEFAULT NULL,
                             language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Table 18–11 on page 18-22 describes the parameters for the CONFIG_UI_FOR_ITEM function.

Table 18–11 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.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
ui_type	varchar2	in / out	This is the type of UI sought and found for each product. Values are 'APPLET' or 'DHTML'.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

Results

This function returns the user interface ID associated with the selected publication.

CONFIG_UI_FOR_ITEM_LF

This function does the same work as CONFIG_UI_FOR_ITEM, but also returns the look_and_feel of the UI ('APPLET', 'BLAF', or 'FORMS').

Considerations Before Running

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:

```
FUNCTION config_ui_for_item_lf ( inventory_item_id IN NUMBER,
                                 organization_id IN NUMBER,
                                 config_lookup_date IN DATE,
                                 ui_type IN OUT VARCHAR2,
                                 calling_application_id IN NUMBER,
                                 usage_name IN VARCHAR2,
                                 look_and_feel OUT VARCHAR2,
                                 publication_mode IN VARCHAR2 DEFAULT NULL,
                                 language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Table 18–12 on page 18-25 describes the parameters for the CONFIG_UI_FOR_ITEM_LF function.

Table 18–12 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
ui_type	varchar2	in / out	This is the type of UI sought and found for each product. Values are 'APPLET' or 'DHTML'.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

Results

This function returns the user interface ID of the selected publication.

CONFIG_UI_FOR_PRODUCT

This function finds UI for a product, 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

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:

```
FUNCTION config_ui_for_product ( product_key IN VARCHAR2,
                                 config_lookup_date IN DATE,
                                 ui_type IN OUT VARCHAR2,
                                 calling_application_id IN NUMBER,
                                 usage_name IN VARCHAR2,
                                 publication_mode IN VARCHAR2 DEFAULT NULL,
                                 language IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Table 18–13, " Parameters for the CONFIG_UI_FOR_PRODUCT Function" describes the parameters for the CONFIG_UI_FOR_PRODUCT function.

Table 18–13 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
ui_type	varchar2	in / out	This is the type of UI sought and found for each product. Values are 'APPLET' or 'DHTML'.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

Results

This function finds UI for a product, 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.

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

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:

```
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 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;
```

Table 18–14, " Parameters for the CONFIG_UIS_FOR_ITEMS Function" describes the parameters for the CONFIG_UIS_FOR_ITEMS function.

Table 18–14 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
ui_type	varchar2_tbl_type	in/ out	List of the types of UIs sought and found for each product. Values are 'APPLET' or 'DHTML'.
calling_application_id	number_tbl_type	in	List of registered IDs of applications for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2_tbl_type	in	List of Usage names to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2_tbl_type	in	List of publication modes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2_tbl_type	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

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

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:

```
FUNCTION config_uis_for_products ( product_key IN VARCHAR2_TBL_TYPE,
                                   config_lookup_date IN DATE_TBL_TYPE,
                                   ui_type IN OUT 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;
```

Table 18–15, " Parameters for the CONFIG_UIS_FOR_PRODUCTS Function" describes the parameters for the CONFIG_UIS_FOR_PRODUCTS function.

Table 18–15 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
config_lookup_date	date_tbl_type,	in	List of dates to search for inside the applicable range for the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
ui_type	varchar2_tbl_type,	in / out	List of the types of UIs sought and found for each product. Values are 'APPLET' or 'DHTML'.
calling_application_id	number_tbl_type,	in	List of registered IDs of applications for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2_tbl_type,	in	List of Usage names to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2_tbl_type,	in	List of publication modes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2_tbl_type	in	List of language codes to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

COPY_CONFIGURATION

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

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, 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.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE copy_configuration( config_hdr_id       IN  NUMBER,
                              config_rev_nbr      IN  NUMBER,
                              new_config_flag     IN  VARCHAR2,
                              out_config_hdr_id   IN OUT NUMBER,
                              out_config_rev_nbr  IN OUT NUMBER,
                              Error_message       IN OUT VARCHAR2,
                              Return_value        IN OUT NUMBER,
                              handle_deleted_flag IN  VARCHAR2 DEFAULT NULL,
                              new_name            IN  VARCHAR2 DEFAULT NULL);
```

Table 18–16 on page 18-34 describes the parameters for the COPY_CONFIGURATION procedure.

Table 18–16 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, and CZ_CONFIG_MESSAGES.
config_rev_nbr	number	in	Specifies which configuration to copy. Uses CZ_CONFIG_HDRS, CZ_CONFIG_INPUTS, CZ_CONFIG_ITEMS, and CZ_CONFIG_MESSAGES.
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.
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

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.

COPY_CONFIGURATION_AUTO

This procedure runs COPY_CONFIGURATION 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" on page 18-33.

Considerations Before Running

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, and messages are copied.

Warnings

If configuration does not exist, or if the copy fails, return_value will be zero, and error_message will contain error information.

Syntax and Parameters

The syntax for this procedure is:

```
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 NUMBER,
                                  out_config_rev_nbr  IN OUT NUMBER,
                                  Error_message       IN OUT VARCHAR2,
                                  Return_value        IN OUT NUMBER,
                                  handle_deleted_flag IN  VARCHAR2 DEFAULT NULL,
                                  new_name           IN  VARCHAR2 DEFAULT NULL);
```

Table 18–17 on page 18-37 describes the parameters for the COPY_CONFIGURATION_AUTO procedure.

Table 18–17 Parameters for the COPY_CONFIGURATION_AUTO Procedure

Parameter	Data Type	Mode	Note
config_hdr_id	number	in	See corresponding parameter in Table 18–16 on page 18-34.
config_rev_nbr	number	in	See corresponding parameter in Table 18–16 on page 18-34.
new_config_flag	varchar2	in	See corresponding parameter in Table 18–16 on page 18-34.

Table 18–17 (Cont.) Parameters for the COPY_CONFIGURATION_AUTO Procedure

Parameter	Data Type	Mode	Note
out_config_hdr_id	number	in/out	See corresponding parameter in Table 18–16 on page 18-34.
out_config_rev_nbr	number	in/out	See corresponding parameter in Table 18–16 on page 18-34.
error_message	varchar2	in/out	See corresponding parameter in Table 18–16 on page 18-34.
return_value	number	in/out	See corresponding parameter in Table 18–16 on page 18-34.
handle_deleted_flag	varchar2 default null	in	See corresponding parameter in Table 18–16 on page 18-34.
new_name	varchar2 default null	in	See corresponding parameter in Table 18–16 on page 18-34.

Considerations After Running

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

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:

```
PROCEDURE DEFAULT_NEW_CFG_DATES( p_creation_date IN OUT DATE,
                                 p_lookup_date IN OUT DATE,
                                 p_effective_date IN OUT DATE);
```

Table 18–18, " Parameters for the DEFAULT_NEW_CFG_DATES Procedure" describes the parameters for the DEFAULT_NEW_CFG_DATES procedure.

Table 18–18 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.

Table 18–18 (Cont.) Parameters for the DEFAULT_NEW_CFG_DATES Procedure

Parameter	Data Type	Mode	Note
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

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. Default date values are determined differently for a restored configuration that for a new configuration.

Considerations Before Running

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:

```
PROCEDURE DEFAULT_RESTORED_CFG_DATES( p_config_hdr_id IN NUMBER,
                                      p_config_rev_nbr IN NUMBER,
                                      p_creation_date IN OUT DATE,
                                      p_lookup_date IN OUT DATE,
                                      p_effective_date IN OUT DATE );
```

Table 18–19, " Parameters for the DEFAULT_RESTORED_CFG_DATES Procedure" describes the parameters for the DEFAULT_RESTORED_CFG_DATES procedure.

Table 18–19 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, it will be returned as is. Otherwise, the existing setting for this configuration is returned.
p_lookup_date	date	in / out	If this is not null, it will be returned as is. Otherwise, the existing setting for this configuration is returned.
p_effective_date	date	in / out	If this is not null, it will be returned as is. Otherwise, the existing setting for this configuration is returned.

Considerations After Running

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

Prerequisites

The configuration to be deleted must exist.

Timing

This procedure should be used when a configuration is obsolete.

Warnings

You should not delete configurations that are referred to by any host applications.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE delete_configuration( config_hdr_id  IN  NUMBER,
                                config_rev_nbr IN  NUMBER,
                                usage_exists   IN OUT NUMBER,
                                Error_message  IN OUT VARCHAR2,
                                Return_value   IN OUT NUMBER);
```

Table 18–20, " Parameters for the DELETE_CONFIGURATION Procedure" on page 18-43 describes the parameters for the DELETE_CONFIGURATION procedure.

Table 18–20 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

Table 18–20 (Cont.) Parameters for the DELETE_CONFIGURATION Procedure

Parameter	Data Type	Mode	Note
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, 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

Troubleshooting

Examine the output in the `error_message` parameter.

ICX_SESSION_TICKET

This function returns a value for the session ticket that Oracle Applications should pass as "icx_session_ticket" when calling Oracle Configurator. This 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.

Considerations Before Running

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:

```
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

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 with usage_name equal to "Any Usage" and publication_mode equal to 'P'.

Considerations Before Running

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:

```
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;
```

Table 18–21, " Parameters for the MODEL_FOR_ITEM Function" on page 18-48 describes the parameters for the MODEL_FOR_ITEM function.

Table 18–21 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, 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_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 calling application.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

Considerations After Running

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

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:

```
FUNCTION model_for_publication_id (publication_id NUMBER)
RETURN NUMBER;
```

Table 18–22, " Parameters for the MODEL_FOR_PUBLICATION_ID Function" on page 18-49 describes the parameters for the MODEL_FOR_PUBLICATION_ID function.

Table 18–22 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.

PUBLICATION_FOR_ITEM

This function returns the publication ID for a specified inventory item.

Considerations Before Running

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:

```
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;
```

Table 18–23, " Parameters for the PUBLICATION_FOR_ITEM Function" on page 18-51 describes the parameters for the PUBLICATION_FOR_ITEM function.

Table 18–23 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, 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

PUBLICATION_FOR_PRODUCT

This function returns the publication ID for a product key.

Considerations Before Running

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:

```
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;
```

Table 18–24, " Parameters for the PUBLICATION_FOR_PRODUCT Function" on page 18-53 describes the parameters for the PUBLICATION_FOR_PRODUCT function.

Table 18–24 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
config_lookup_date	date	in	Date to search for inside the applicable range for the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number	in	The registered ID of an application for which the Model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

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

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:

```
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;
```

Table 18–25, " Parameters for the PUBLICATION_FOR_SAVED_CONFIG Function" on page 18-55 describes the parameters for the PUBLICATION_FOR_SAVED_CONFIG function.

Table 18–25 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 Section 18.2.6.2, "Applicability Parameters" on page 18-6.
calling_application_id	number	in	The registered ID of an application for which the model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
usage_name	varchar2	in	Usage name to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
publication_mode	varchar2	in	Publication mode to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.
language	varchar2	in	Language code to search for in the publication. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

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 with usage_name equal to "Any Usage" and publication_mode equal to 'P'.

Considerations Before Running

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:

```
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;
```

Table 18–26, " Parameters for the UI_FOR_ITEM Function" on page 18-57 describes the parameters for the UI_FOR_ITEM function.

Table 18–26 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, 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_creation_date	date	in	This is the date the configuration was created.
ui_type	varchar2	in	This is the type of UI sought and found for each product. Values are 'APPLET' or 'DHTML'.
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 calling application.
calling_application_id	number	in	The registered ID of an application for which the model is published. See Section 18.2.6.2, "Applicability Parameters" on page 18-6.

UI_FOR_PUBLICATION_ID

This function returns a UI definition (`ui_def_id`) for a specified publication ID.

Considerations Before Running

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:

```
FUNCTION ui_for_publication_id ( publication_id NUMBER )
RETURN NUMBER;
```

Table 18–27, " Parameters for the UI_FOR_PUBLICATION_ID Function" on page 18-59 describes the parameters for the UI_FOR_PUBLICATION_ID function. See Example 18–1 on page 18-59 for an example of how these parameters are used.

Table 18–27 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 executed 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.

Example 18–1 Using the UI_FOR_PUBLICATION_ID Function

```
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

This procedure is a single call validation procedure that uses tables to exchange multi-valued data. A validation_status and a table of XML messages are returned.

Considerations Before Running

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE VALIDATE ( config_input_list IN CFG_INPUT_LIST,
                     init_message IN VARCHAR2,
                     config_messages IN OUT CFG_OUTPUT_PIECES,
                     validation_status IN OUT NUMBER,
                     URL IN VARCHAR2 DEFAULT FND_PROFILE.Value('CZ_UIMGR_URL'),
                     p_validation_type IN VARCHAR2 DEFAULT CZ_API_PUB.VALIDATE_
ORDER));
```

Table 18–28, " Parameters for the VALIDATE Procedure" on page 18-61 describes the parameters for the VALIDATE procedure.

Table 18–28 Parameters for the VALIDATE Procedure

Parameter	Data Type	Mode	Note
config_input_list	CFG_INPUT_LIST[1]	in	This is a list of input selections.
init_message	varchar2	in	Initialization message
config_messages	CFG_OUTPUT_PIECES[2]	out	This is a table of the output XML messages produced by validating the configuration.

Table 18–28 (Cont.) Parameters for the VALIDATE Procedure

Parameter	Data Type	Mode	Note
validation_status	varchar2	out	The status code returned by validating the configuration: 0 - CONFIG_PROCESSED 1 - CONFIG_PROCESSED_NO_TERMINATE 2 - INIT_TOO_LONG 3 - INVALID_OPTION_REQUEST 4 - CONFIG_EXCEPTION 5 - DATABASE_ERROR 6 - UTL_HTTP_INIT_FAILED 7 - UTL_HTTP_REQUEST_FAILED
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.

[1] See Section 18.3.1, "Custom Data Types" on page 18-9 for a definition of this type.

[2] See Section 18.3.1, "Custom Data Types" on page 18-9 for a definition of this type.

Example

For an example of how these parameters are used, see Section 11.3, "Calling the CZ_CF_API.VALIDATE Procedure" on page 11-4.

Considerations After Running

Results

This procedure returns the following values:

Return Value	Description
CONFIG_PROCESSED	Configuration processed successfully, and a termination message was returned.
CONFIG_PROCESSED_NO_TERMINATE	Configuration, no termination message 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	Procedure uses UTL_HTTP package to pass data to Configurator Servlet. These exceptions can be returned by UTL_HTTP procedures. See *Oracle8i Supplied PL/SQL Packages Reference* for additional information.
UTL_HTTP_REQUEST_FAILED	

19

Programmatic Tools for Maintenance

This chapter describes a set of programmatic tools that you can use primarily to maintain a deployed runtime Oracle Configurator.

For information on tools for developing a configuration model or deploying a runtime Oracle Configurator, see Chapter 18, "Programmatic Tools for Development".

19.1 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.

19.1.1 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
- Generation and refreshing of Active Models and User Interfaces
- Publication of Active Models and User Interfaces
- Initial execution and refreshing of Item Master Populators

19.1.2 Installation of the Package

The information provided for the package CZ_CF_API in Section 18.1.3, "Installation of the Package" on page 18-3 also applies to the package CZ_modelOperations_pub.

19.1.3 References for Working with PL/SQL Procedures

For background information and details on basic aspects of working with the PL/SQL procedures in this package, see Table 18–2 on page 18-4 in Section 18.1.4, "References for Working with PL/SQL Procedures and Functions", which suggests relevant topics in the Oracle Documentation Library.

19.2 Choosing the Right Tool for the Job

These procedures are described in detail in Section 19.4.3, "Procedures in the CZ_modelOperations_pub Package" on page 19-9.

19.2.1 Importing and Refreshing Models

Use these procedures to import or refresh Models:

- IMPORT_SINGLE_BILL
- REFRESH_SINGLE_MODEL

19.2.2 Generating the Active Model

Use this procedure to generate the Active Model:

- GENERATE_LOGIC

19.2.3 Generating and Refreshing UIs

Use these procedures to generate or refresh a user interface:

- CREATE_UI
- REFRESH_UI

19.2.4 Copying Models

Use this procedure to make a deep copy of a specified Model:

- DEEP_MODEL_COPY

19.2.5 Publishing Models

Use these procedures to publish or republish Models:

- PUBLISH_MODEL
- REPUBLISH_MODEL

19.2.6 Running Populators

Use these procedures to run Populators:

- EXECUTE_POPULATOR
- REPOPULATE

19.3 Queries to Support the CZ_modelOperations_pub Package

This section contains PL/SQL queries which provide the values that you need to provide as parameters to certain procedures in the CZ_modelOperations_pub package.

19.3.1 Querying for Model IDs

Example 19–1 on page 19-3 provides a PL/SQL query that lists the names and IDs of source (not published) Models, and the Folders which 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. You would use this ID as a value for the parameter `p_devl_project_id` to the following procedures:

- CREATE_UI
- GENERATE_LOGIC
- REFRESH_SINGLE_MODEL
- DEEP_MODEL_COPY
- REPOPULATE

Example 19–1 Query for Models and Folders

```
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.

```
  P.name like '%your Model's name%' and
```

19.3.2 Querying for User Interface IDs

Example 19–2 on page 19-4 provides a PL/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, you would use the query in Example 19–1 on page 19-3.

This query selects values for the column CZ_UI_DEFS.UI_DEF_ID. This UI_DEF_ID is returned by the procedure CREATE_UI. You would use this ID as a value for the p_ui_def_id parameter for the procedure REFRESH_UI.

Example 19–2 Query for User Interface IDs

```
select
  ui_def_id,
  name
from
  cz_ui_defs
where
  devl_project_id = devl_project_ID
and
  deleted_flag = '0';
```

19.3.3 Querying for Referenced User Interface IDs

Example 19–3 on page 19-5 provides a PL/SQL query that lists the IDs of available referenced (child) user interfaces for a specified *parent_ui_def_ID*. To determine the *parent_ui_def_ID* for a specified Model, you would use the query in Example 19–2 on page 19-4.

This query selects a value for the column CZ_UI_NODES.UI_DEF_ID. You would then use this value as a parameter for the following procedures:

- REFRESH_UI

Example 19–3 Query for Referenced User Interface IDs

```
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;
```

19.3.4 Querying for Populators

Example 19–4 on page 19-5 provides a PL/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, you would use the query in Example 19–1 on page 19-3.

This query selects a value for the column CZ_POPULATORS.POPULATOR_ID. You would use this value as a parameter for the following procedures:

- EXECUTE_POPULATOR

Example 19–4 Query for Populators

```
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';
```

19.3.5 Querying for Error and Warning Information

Example 19–5 on page 19-6 provides a PL / 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
- GENERATE_LOGIC
- IMPORT_SINGLE_BILL

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 procedure. The URGENCY values used by GENERATE_LOGIC 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.

Example 19–5 Query for Error and Warning Information

```
select
  urgency,
  statuscode,
  message
from
  cz_db_logs
where
  run_id = run_ID_returned_from_procedure;
```

19.4 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 Table 19–1 on page 19-9.
- Descriptions of the custom data types defined in the package are also provided, in Custom Data Types on page 19-7.
- For a basic example of how to call one of the functions in the CZ_CF_API package, see Example 19–6, "Using the GENERATE_LOGIC Procedure" on page 19-18.
- See also Section 19.1, "Overview of the CZ_modelOperations_pub Package" on page 19-1.

19.4.1 Custom Data Types

There are no custom data types defined in the CZ_modelOperations_pub package.

19.4.2 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.

19.4.2.1 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. Examples:

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.

19.4.2.2 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
```

19.4.2.3 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:

```
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 Example 19–5 on page 19-6.

19.4.3 Procedures in the CZ_modelOperations_pub Package

This section provides descriptions of each of the procedures in the CZ_modelOperations_pub package, arranged alphabetically. These procedures are listed in Table 19–1 on page 19-9.

Table 19–1 Procedures in the Package CZ_modelOperations_pub

API Name
CREATE_UI on page 19-10
DEEP_MODEL_COPY on page 19-13
EXECUTE_POPULATOR on page 19-15
GENERATE_LOGIC on page 19-17
IMPORT_SINGLE_BILL on page 19-19
PUBLISH_MODEL on page 19-20
REFRESH_SINGLE_MODEL on page 19-21
REFRESH_UI on page 19-22
REPOPULATE on page 19-24
REPUBLISH_MODEL on page 19-26

CREATE_UI

The CREATE_UI procedure generates a new user interface for a model.

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.

Considerations Before Running

Alternatives

As an alternative to using this procedure, you can create a UI in Oracle Configurator Developer, by switching to the UI module, selecting the root UI node, then choosing **Create** > **New User Interface**.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE create_ui(p_api_version IN NUMBER,
                    p_devl_project_id  IN  NUMBER,
                    x_ui_def_id OUT NUMBER,
                    x_run_id OUT NUMBER,
                    x_status OUT 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');
```

Table 19–2 on page 19-11 describes the parameters for the CREATE_UI procedure.

Table 19–2 Parameters for the CREATE_UI Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_devl_project_id	in	number	The ID of the Model for which to create a UI. See Example 19–1 on page 19-3 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, 0 is stored.
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 "display in 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 display in UI". If this parameter is '0' then the new UI will respect the "display in 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 is 'COMPONENTS'. The default is 'BLAF'.

Table 19–2 (Cont.) Parameters for the CREATE_UI Procedure

Parameter	Mode	Data Type	Note
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.
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'.

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.

Syntax and Parameters

The syntax for this procedure is:

```
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 NUMBER,
                          x_run_id      OUT NUMBER,
                          x_status      OUT NUMBER);
```

Table 19–3 on page 19-13 describes the parameters for the DEEP_MODEL_COPY procedure.

Table 19–3 Parameters for the DEEP_MODEL_COPY Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_devl_project_id	in	number	The ID of the Model of which a copy is to be made. See Example 19–1 on page 19-3 for a query that provides this ID (DEVL_PROJECT_ID).
p_folder	in	number	The folder to which the copy is made. See Example 19–1 on page 19-3 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 as well.

Table 19–3 (Cont.) Parameters for the DEEP_MODEL_COPY Procedure

Parameter	Mode	Data Type	Note
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 executing 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 Oracle Configurator schema describes the structure of the Active Model.

See the description of REPOPULATE on page 19-24 for information on the related procedure for repopulating Model structure.

Considerations Before Running

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.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE execute_populator(p_api_version IN NUMBER,
                            p_populator_id IN NUMBER,
                            p_imp_run_id IN OUT VARCHAR2,
                            x_run_id OUT NUMBER,
                            x_status OUT NUMBER);
```

Table 19–4 on page 19-15 describes the parameters for the EXECUTE_POPULATOR procedure.

Table 19–4 Parameters for the EXECUTE_POPULATOR Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.

Table 19–4 (Cont.) Parameters for the EXECUTE_POPULATOR Procedure

Parameter	Mode	Data Type	Note
p_populator_id	in	number	The value of CZ_POPULATORS.POPULATOR_ID for the Populator to be executed.
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, 0 is stored.
x_status	out	number	Either G_STATUS_ERROR or G_STATUS_SUCCESS.

GENERATE_LOGIC

The GENERATE_LOGIC procedure generates the Active Model for a Model and all of its referenced Models if necessary. Generating the Active Model is sometimes referred to as "generating logic".

Considerations Before Running

Alternatives

As an alternative to using this procedure, you can generate logic in Oracle Configurator Developer, by choosing Tools > Generate Active Model.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE generate_logic(p_api_version IN NUMBER,
                         p_devl_project_id IN  NUMBER,
                         x_run_id OUT NUMBER,
                         x_status OUT NUMBER);
```

Table 19–5 on page 19-17 describes the parameters for the GENERATE_LOGIC procedure.

Table 19–5 Parameters for the GENERATE_LOGIC Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_devl_project_id	in	number	The ID of the Model for which to generate logic. See Example 19–1 on page 19-3 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, 0 is stored.
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_WARNING, or G_STATUS_SUCCESS.

Example

When executed 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.

Example 19–6 Using the GENERATE_LOGIC Procedure

```
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).

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE import_single_bill(p_api_version IN  NUMBER,
                             p_org_id IN NUMBER,
                             p_top_inv_item_id IN NUMBER,
                             x_run_info_id OUT NUMBER,
                             x_run_id OUT NUMBER,
                             x_status OUT NUMBER);
```

Table 19–6 on page 19-19 describes the parameters for the IMPORT_SINGLE_BILL procedure.

Table 19–6 Parameters for the IMPORT_SINGLE_BILL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_org_id	in	number	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_info_id	out	number	This is *not* CZ_DB_LOGS.RUN_ID, but is generated by executing: `(select max(run_id) from CZ_XFR_RUN_INFOS);`
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.

PUBLISH_MODEL

After a publication record is created through Oracle Configurator Developer, the PUBLISH_MODEL procedure will export the models and UIs associated with the publication.

Considerations Before Running

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 using the Model Publishing window.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE publish_model(p_api_version IN  NUMBER,
                        p_publication_id IN NUMBER,
                        x_run_id OUT NUMBER,
                        x_status OUT NUMBER);
```

Table 19–7 on page 19-20 describes the parameters for the PUBLISH_MODEL procedure.

Table 19–7 Parameters for the PUBLISH_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
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.

REFRESH_SINGLE_MODEL

The REFRESH_SINGLE_MODEL procedure can be used to refresh a model imported from Oracle Bills of Materials (BOM).

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE refresh_single_model(p_api_version      IN  NUMBER,
                               p_devl_project_id  IN  VARCHAR2,
                               x_run_info_id      OUT NUMBER,
                               x_run_id           OUT NUMBER,
                               x_status           OUT NUMBER);
```

Table 19–8 on page 19-21 describes the parameters for the REFRESH_SINGLE_MODEL procedure.

Table 19–8 Parameters for the REFRESH_SINGLE_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_devl_project_id	in	varchar2	The ID of the Model for which to refresh imported data. See Example 19–1 on page 19-3 for a query that provides this ID (DEVL_PROJECT_ID).
x_run_info_id	out	number	This is *not* CZ_DB_LOGS.RUN_ID, but is generated by executing: `(select max(run_id) from CZ_XFR_RUN_INFOS);`
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.

Considerations Before Running

Restrictions and Limitations

This procedure only refreshes the UI specified. It does not refresh referenced user interfaces.

Alternatives

As an alternative to using this procedure, you can refresh a UI in Oracle Configurator Developer, by switching to the UI module, selecting a User Interface, then choosing Edit > Refresh.

Syntax and Parameters

The syntax for this procedure is:

```
PROCEDURE refresh_ui(p_api_version IN     NUMBER,
                     p_ui_def_id   IN OUT NUMBER,
                     x_run_id OUT NUMBER,
                     x_status OUT NUMBER);
```

Table 19–9 on page 19-22 describes the parameters for the REFRESH_UI procedure.

Table 19–9 Parameters for the REFRESH_UI Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_ui_def_id	in/out	number	UI definition ID of user interface to be refreshed. If user interface is Applet style, a new ui_def_id is returned through this parameter. If the style is DHTML, 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, 0 is stored.

Table 19–9 (Cont.) Parameters for the REFRESH_UI Procedure

Parameter	Mode	Data Type	Note
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_WARNING or G_STATUS_SUCCESS.

REPOPULATE

The REPOPULATE procedure iterates through all Populators associated with the input model and executes them.

Considerations Before Running

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, by choosing Tools > RePopulate.

Syntax and Parameters

The syntax for this procedure is:

```
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 NUMBER,
                     x_status OUT NUMBER);
```

Table 19–10 on page 19-24 describes the parameters for the REPOPULATE procedure.

Table 19–10 Parameters for the REPOPULATE Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	See API Version Numbers on page 19-7.
p_devl_project_id	in	number	The ID of the Model to repopulate. See Example 19–1 on page 19-3 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.

Table 19–10 (Cont.) Parameters for the REPOPULATE Procedure

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, 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

Alternatives

As an alternative to using this procedure, you can republish an existing model in Oracle Configurator Developer, by choosing Tools > Publishing..., selecting a model >Republish.

Syntax and Parameters

The syntax for this procedure is:

```
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 NUMBER,
                    x_status OUT NUMBER);
```

Table 19–11 on page 19-27 describes the parameters for the REPUBLISH_MODEL procedure.

Table 19–11 Parameters for the REPUBLISH_MODEL Procedure

Parameter	Mode	Data Type	Note
p_api_version	in	number	1.0. See API Version Numbers on page 19-7 for additional information.
p_publication_id	in	number	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.
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, 0 is stored.
x_status	out	number	Either G_STATUS_ERROR, G_STATUS_ SUCCESS, or G_STATUS_WARNING.

Part V

Runtime Configurator

Part V presents information for deploying a runtime Oracle Configurator that is embedded in a host Oracle Application or a custom hosting application as described in Section 1.5, "Deployment Tasks" on page 1-7. Part V contains the following chapters:

- Chapter 20, "User Interface Deployment"
- Chapter 21, "Deployment Considerations"
- Chapter 22, "Managing Configurations"

20

User Interface Deployment

Deployment involves making a runtime Oracle Configurator available to end users.

This chapter and Section 21, "Deployment Considerations" on page 21-1 describe activities required to complete deployment of a runtime Oracle Configurator embedded in a host Oracle Application such as Order Management or *i*Store.

See Section 3.1, "Database Uses" on page 3-1 for an overview of possible deployment environments and architecture.

20.1 Calling an Embedded Oracle Configurator

Oracle Applications uses an internet server, such as Oracle Internet Application Server (*i*AS), to run the Oracle Configurator Servlet. The Oracle Configurator Servlet connects the runtime Oracle Configurator's URL to the Oracle Configurator schema. That URL is set in 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.

Oracle Configurator embedded in Oracle Applications uses one of these user interfaces:

- a non-customizable window produced by a Java applet
- a customizable Web browser window produced by DHTML (Dynamic HTML)

HTML-based applications, such as *i*Store or a custom Web application, can only use the DHTML interface.

20.1.1 Java applet Requirements

The requirements for using the Java applet interface are already in place when launching the runtime Oracle Configurator from Oracle Applications:

- JInitiator version 1.1.8.7 or later must be installed on the client machine (Oracle Configurator also supports version 1.3.1.9). The JInitiator properties should set Network Access to "Unrestricted" and the Java Runtime Parameters to:

  ```
  -mx128m -Dcache.size=50000000
  ```

- Pricing behavior must be set for item price display type and price data update method. See Chapter 13, "Pricing and ATP in Oracle Configurator" on page 13-1.

20.1.2 DHTML Requirements

The requirements for the runtime Oracle Configurator to use the DHTML interface are:

- Stylesheets and JavaScript must be enabled in the browser running the DHTML runtime Oracle Configurator. Stylesheets and JavaScript are essential components of DHTML.
- Use Microsoft Internet Explorer 4.0 or higher, or Netscape Navigator 4.07 or higher, to best view the DHTML runtime Oracle Configurator. The browser must support frames.
- The browser must be set up to accept and / or send cookies.
- Recommended screen resolution is 800 X 600 or higher. 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.
- Pricing behavior must be set for item price display type and price data update method. See Section 13.1, "Pricing in a Runtime Oracle Configurator" on page 13-1. Table 20–1, " Information Details Mapped to Oracle Configurator Documentation" on page 20-2 lists Oracle Configurator Documentation for more information:

Table 20–1 Information Details Mapped to Oracle Configurator Documentation

For details on ...	See ...
The initialization and termination messages, and other elements of a custom Web applications	Chapter 9, "Session Initialization"

Table 20–1 (Cont.) Information Details Mapped to Oracle Configurator Documentation

For details on ...	See ...
Setting up the OC Servlet	*Oracle Configurator Installation Guide*
Using Oracle Configurator Developer	*Oracle Configurator Developer User's Guide*

20.1.3 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 corressponding actions at runtime, see the *Oracle Configurator Developer User's Guide*.

21

Deployment Considerations

This chapter and Section 20, "User Interface Deployment" on page 20-1 describe activities required to complete deployment of a runtime Oracle Configurator embedded in a host Oracle Application such as Order Management or *i*Store.

See Section 3.1, "Database Uses" on page 3-1 for an overview of possible deployment environments and architecture.

21.1 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:

- Section 21.1.1, "Architectural Considerations" on page 21-1
- Section 21.1.2, "Server Considerations" on page 21-2
- Section 21.1.4, "Implementing Oracle Configurator with Secure Sockets Layer" on page 21-5
- Section 21.1.5, "Network Considerations" on page 21-9
- Section 21.1.6, "Multiple Language Support Considerations" on page 21-10

While it is not possible to provide an exact prescription for your particular application, this section describes the principles that affect a typical Oracle Configurator deployment.

21.1.1 Architectural Considerations

The architecture of an application often determines the limits within which the other factors must operate. If you lay out your sequence of pages and controls in an

inefficient way, then it will be difficult to overcome this inefficiency later simply by tuning your server software or augmenting your hardware.

As described in Section 21.1.2, "Server Considerations" on page 21-2, an important tuning factor is the number of end users conducting configuration sessions on each instance of the servlet engine. For *i*AS, the servlet engine is Apache JServ.

Model loading and data access depend on how the application was written. To get the information required to start tuning your servlet engine 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 can select page by page
- How users can navigate from page to page
- What selectable items are on a page (such as the number of Features per Component, which should generally not exceed twenty)
- When events might be interrupted (for example, when a user pauses a long time to consider his choices, or turns to another task before returning to make a selection)

21.1.2 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. For information about factors that have an impact on your resources, see

Section 21.1.1, "Architectural Considerations" on page 21-1 and the section on load balancing in the *Oracle Configurator Installation Guide*.

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 Functional Companions used by the application
- The number of CPUs
- The memory per CPU

  The JDK uses about 16 MB. The JVM for each JServ uses about 45 MB. Oracle Configurator uses native threads.
- The number of JServ instances running
- The number of connections available in the connection pool (see Section 21.1.2.1, "Connection Pooling" on page 21-4)
- The frequency and type of database accesses caused by user actions (see the section on load balancing in the *Oracle Configurator Installation Guide*)

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.

21.1.2.1 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—then a connection is obtained from the pool. When the session is over, the connection is returned to the pool. Each connection uses up 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.

21.1.3 Establishing End User Access

End users of the runtime Oracle Configurator (DHTML or Java applet) are established through Oracle Applications administration and reside in the Oracle Applications database.

End users click on a Configure button or a menu command in Oracle Applications to call the runtime Oracle Configurator. This action causes the calling application to initialize a runtime Oracle Configurator session with an initialization message. When an end user saves a configuration and exits a runtime Oracle Configurator session, the runtime Oracle Configurator sends the calling application a termination message to terminate the runtime Oracle Configurator session.

For more information about the initialization message, see Chapter 9, "Session Initialization". 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 machine where the hosting application is running. For more information about publication applicability parameters, see Section 17.2.3 on page 17-8 and the *Oracle Configurator Developer User's Guide*.

21.1.4 Implementing Oracle Configurator with Secure Sockets Layer

Secure Sockets Layer (SSL) is a protocol that creates a secure connection between a client and a server machine and enables you to safely transmit private documents over the Internet. SSL uses a public **key** to encrypt data that is transferred over the SSL connection. (A key is a password or table needed to decipher encoded data.)

To set up Oracle Configurator to run in SSL mode, perform the following:

1. Install Internet Application Server (iAS) version 1.0.2.2 or later and configure it to run SSL.

   For more information, consult the *Apache 1.3 User's Guide* and the Apache-SSL Web site at `http://www.apache-ssl.org`.

2. Install Oracle Configurator version 16-38 or later. (To upgrade to version 16-38, apply patch 2102442.) See the *Oracle Configurator Installation Guide*.

3. Edit the Oracle Configurator system parameters in `jserv.properties` to use the SSL server and port. This step is required for SSL to support HTML hosting applications such as *i*Store. See Section 21.1.4.1, "Editing jserv.properties for Secure Sockets Layer" on page 21-6.

4. Verify that AltBatchValidateURL is present and set correctly in the ORAAPPS_INTEGRATE section of the CZ_DB_SETTINGS table. See Section 21.1.4.2, "Verifying AltBatchValidateURL" on page 21-6.

5. Set the profile option BOM:Configurator URL of UI Manager to the *secure* URL. This option should be set at the Site level, but it can also be set at the User level. For information about setting profile options, see the *Oracle Applications User's Guide*.

   You can test this URL using the same method described in Section 21.1.4.2, "Verifying AltBatchValidateURL" on page 21-6.

6. Set up your Web browser and Jinitiator to support SSL. See Section 21.1.4.3, "Enabling the Oracle Configurator Client for Secure Sockets Layer" on page 21-8.

21.1.4.1 Editing jserv.properties for Secure Sockets Layer

To support SSL, all of the Oracle Configurator system parameters in the jserv.properties file must use the secure "https" protocol, and must include the name and port number of your secure server. (All URLs that require an SSL connection start with "https" instead of "http".)

In Example 21–1, `myservername` is the name of the *i*AS server and `443` is the secure server port number.

Example 21–1 The jserv.properties File System Parameters for SSL

```
wrapper.bin.parameters=-Dcz.uiservlet.templateURL=https://myse
rvername.com:443/OA_HTML/US/czFraNS.htm

wrapper.bin.parameters=-Dcz.uiservlet.URL=https://myservername
.com:443/configurator/oracle.apps.cz.servlet.UiServlet

wrapper.bin.parameters=-Dcz.uiservlet.proxyscript=https://myse
rvername.com:443/OA_HTML/czProxy.js

wrapper.bin.parameters=-Dcz.html.source.treeview=https://myser
vername.com:443/OA_HTML/cztree.jsp

wrapper.bin.parameters=-Dcz.html.source.display=https://myserv
ername.com:443/OA_HTML/czdisp.jsp

wrapper.bin.parameters=-Dcz.uiservlet.blaftemplateURL=https://
myservername.com:443/OA_HTML/US/czBlafTemplate.jsp

wrapper.bin.parameters=-Dcz.uiservlet.formtemplateURL=https://
myservername.com:443/OA_HTML/US/czFormTemplate.jsp

wrapper.bin.parameters=-Dcz.html.source.formtreeview=https://m
yservername.com:443/OA_HTML/czFormTree.jsp
```

Each parameter should be defined on a single line in jserv.properties.

21.1.4.2 Verifying AltBatchValidateURL

The AltBatchValidateURL setting allows the batch validation process to bypass the URL that would normally be used for batch validation. This is needed if Oracle

Configurator uses SSL. The batch validation process runs, for example, when booking an order in Oracle Order Management.

The value of AltBatchValidateURL must be your non-secure URL. Since the batch validation process communicates between the database and Web server, it is not necessary for this communication to use SSL.

Upgrading or installing Oracle Configurator does not automatically add AltBatchValidateURL to the CZ_DB_SETTINGS table. Therefore, you must add it to CZ_DB_SETTINGS if you are setting up Oracle Configurator to run in SSL mode for the first time.

To insert AltBatchValidateURL into CZ_DB_SETTINGS, use the SQL *Plus statement shown in Example 21–2.

Example 21–2 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/configurator/oracle.apps.cz.servlet.UiServlet',
'Non-secure URL')
```

> **Note:** If AltBatchValidateURL does not exist in the CZ_DB_SETTINGS table, the batch validation process fails because it uses the *secure* URL that is specified by the profile option BOM:Configurator URL of UI Manager.

If you previously set up Oracle Configurator to run in SSL mode and want to determine the value of AltBatchValidateURL, use the SQL *Plus statement shown in Example 21–3.

Example 21–3 Determining the Value of AltBatchValidateURL

```
SELECT * FROM cz_db_settings WHERE setting_id = 'AltBatchValidateURL';
```

The value returned is the same as the Java property cz.uiservlet.url for the non-secure URL. For example:

```
http://servername.com:8808/configurator/oracle.apps.cz.servlet
.UiServlet
```

You can test this URL by entering it in a Web browser followed by "?test=version". The result should be the build and schema version of Oracle Configurator running on the Apache server. For example:

```
http://servername.com:8808/configurator/oracle.apps.cz.servlet
.UiServlet?test=version
```

For more information about cz.uiservlet.url, see the *Oracle Configurator Installation Guide*.

21.1.4.3 Enabling the Oracle Configurator Client for Secure Sockets Layer

You can run an Oracle Configurator in SSL mode using either a DHTML or a Java applet UI, but some setup is required.

21.1.4.3.1 DHTML Setup Because a DHTML UI runs in a Web browser, you must ensure that your browser supports SSL. To enable your browser to support SSL, perform the following:

- In Netscape Navigator, choose **Communicator > Tools > Security Info**, then click **Navigator**. Select from the available options to enable SSL.
- In Internet Explorer, choose **Tools > Internet Options**, then select the **Advanced** tab. Scroll down to view the **Security** section, and select the appropriate options to enable SSL.

21.1.4.3.2 Java applet Setup To use SSL with the Java applet, you must:

- Install JInitiator version 1.1.8.3 or later (prior versions do not support SSL)
- Set up the root certificate

If you know the root certificate, go the JInitiator security directory (for example, C:\Program Files\Oracle\JInitiator 1.1.8.13\lib\security), open the file certdb.txt, and add the root certificate to the end of the file.

If you do not know the root certificate, perform the following:

1. Using Internet Explorer, navigate to your secure URL.
2. Choose **File > Properties**.
3. Click **Certificates**.
4. Select the **Certification Path** tab, then select the top entry for the Certification path.
5. Click **View Certificate**.

6. Select the **Details** tab, then click **Copy to File**.
7. Click **Next**, then choose "Base64 encoded X.509 (.CER)".
8. Click **Next**, then enter a filename to export the certificate (for example, test.cer).
9. Click **Next**, then **Finish**. Close the Certificate and Properties windows by clicking **OK**.
10. Using a text editor, open the file you saved in step 8.
11. Copy the certificate, then paste its contents at the end of the certdb.txt file.
12. Save your work.

Note: It is strongly recommended that you also have your server's SSL certificate signed by one of the following Certificate Authorities (CAs): VeriSign Inc.; RSA Data Security Inc.; GTE Corporation; GTE CyberTrust Solutions Inc. If your certificate is not signed by one of these CAs, you may receive errors when running the Java applet in SSL mode.

21.1.5 Network Considerations

There are a number of network issues that can cause serious problems for your deployment if not handled correctly.

21.1.5.1 Firewalls and Timeouts

It is likely that your application requires more than one server machine. It is recommended that there be separate servers for the Oracle database server and the internet server. If there are firewalls between servers, these firewalls must allow persistent database connections between them.

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, this limit can prevent Oracle Configurator from operating.

21.1.5.2 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 as DHTML) 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, it is very 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.

21.1.5.3 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.

21.1.6 Multiple Language Support Considerations

If you are implementing Multiple Language Support (MLS), see Chapter 14 on page 14-1.

21.1.7 Performance Considerations

For information about improving the performance of your runtime Oracle Configurator, see the *Oracle Configurator Performance Guide*.

22

Managing Configurations

This chapter explains the data structures produced by Oracle Configurator during a configuration session.

- Section 22.1, "About Configurations"
- Section 22.2, "Configuration Identity"

For general information, see Chapter 2, "Configurator Architecture". 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.*

22.1 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 the selections of an end user, which cause configuration rules to be applied against a configuration model. Oracle Configurator validates the selections, resulting in a configuration.

A configuration can be:

- Valid or invalid

  A valid configuration is one in which there are no contradictions to the rules. An invalid configuration contains contradictions.

- Complete or incomplete

  A complete configuration is one in which all the required selections have been made. 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 no selections have been made against the configuration model, and the logic state of its items and options is Unknown.

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. Saving a configuration does not require that it be either valid or complete.

If a configuration has been saved after a configuration session, it can later 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. When the configuration is restored, the restored inputs are reasserted against the model to produce a configuration.
- If the configuration model or rules have changed since the configuration was saved, validation failures may occur as a result of inputs that no longer match the model.
- Since restoring a configuration reasserts all the configuration inputs to the model, restoring a configuration programmatically with the CIO is normally not faster than creating one, 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.

22.2 Configuration Identity

Configurations commonly consist of a single instance of your configuration model and a set of configuration inputs.

Each configuration, once it has been saved during a configuration session, is identified by a Configuration Header ID, which is stored in the CZ schema as CZ_CONFIG_HDRS.CONFIG_HDR_ID.

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 Configurator eTRM on Metalink, Oracle's technical support Web site.

Part VI

Appendices

Part VI contains the following appendices:

- Appendix A, "Common Tasks"
- Appendix B, "Concurrent Programs"
- Appendix C, "CZ Subschemas"
- Appendix D, "Code Examples"

A

Common Tasks

This appendix describes common tasks of an Oracle Configurator implementation:

- Running Configurator Concurrent Programs
- Connecting to a Database Instance
- Verifying Configurator Schema Version
- Server Administration
- Viewing Status of Configurator Concurrent Programs Requests
- Viewing Log Files
- Checking BOM and Configuration Model Similarity

For details about specific Oracle Configurator concurrent programs, see Appendix B, "Concurrent Programs".

A.1 Running Configurator Concurrent Programs

In order to run the Oracle Configurator concurrent programs you must be assigned the appropriate Configurator responsibility for the specific program (either Configurator Administrator or Configurator Developer). For information about assigning responsibilities, see *Installing Oracle Applications*.

The following procedure describes how to run a concurrent program generally. For specifics, see the relevant sections in Appendix B, "Concurrent Programs".

1. Determine in which database instance you must run the concurrent program.
2. Log into Oracle Applications connecting to the appropriate database instance.

3. Select either the Configurator Administrator or Configurator Developer responsibility, depending on which is required for the concurrent program you intend to run.
4. Navigate to the concurrent program by selecting it from the list of values in the Navigator window. Some concurrent programs are grouped in categories. For example, **Configurator Administration** > **View Configurator Parameters.**
5. If relevant for the concurrent program you have selected:
   a. Enter or select the input parameters from a list of values in the Parameters window. You can query valid values by entering the % wildcard.
   b. Click **OK**.

   The parameters for each concurrent program are listed and described in Appendix B, "Concurrent Programs".
6. In the Submit Request window, click **Submit**.
7. If you want to submit another request for the same concurrent program but with different parameters, then click **Yes** in the Decision window.

   For additional information about submitting a request for a concurrent program, see the *Oracle Applications User Guide*.
8. If the concurrent program generates output, warnings, or errors, examine the log file. For more information see Section A.6, "Viewing Log Files" on page A-4.
9. To view the status of your concurrent program, see Section A.5 on page A-4.

A.2 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.

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 Oracle Configurator schema by connecting to the database instance as a user of the schema.

   ```
   Example:

   SQL> connect oc/ocpass@appssid
   ```

where oc is the owner (DBOwner) of the Oracle Configurator schema, and *ocpass* is the owner's password, and appssid is the name for the Oracle8*i* Enterprise Edition instance on which the Oracle Configurator schema is installed.

Alternatively, connect to the database instance as a user with DBA privileges:

Example:

```
SQL> connect dba/dbapass@appssid
```

A.3 Verifying Configurator Schema Version

You can determine the version information of an Oracle Configurator schema by either running the View Configurator Parameters 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:

```
SQL> select setting_id, value, desc_text
from cz_db_settings
where setting_id like '%_VERSION"
```

Querying the version of Release 11*i* available with the publication of this book results in MAJOR_VERSION = 19, MINOR_VERSION = h.

These values will vary depending on the latest installed version. To determine which version of Oracle Configurator Developer goes with which version of Configurator, refer to note #131088.1 on Metalink.

For information about MAJOR_VERSION, MINOR_VERSION, and other CZ_DB_SETTINGS parameters, see Section 4.4 on page 4-7.

A.4 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 Chapter 5, "Populating the CZ Schema")

- Publishing configuration models (see Chapter 17, "Publishing Configuration Models")

Oracle Configurator provides the following Server Administration concurrent programs for the Configurator Administrator responsibility in Oracle Applications:

- Define Remote Server
- Enable Remote Server
- Modify Server Definition
- View Servers

For details on running the above concurrent programs, see Section B.2, "Server Administration Concurrent Programs" on page B-4.

A.5 Viewing Status of Configurator Concurrent Programs Requests

Since all reports, programs, and requests are run as concurrent programs in Oracle Applications, the Requests concurrent program is used to:

- View the status and output of your requests
- View a list of all submitted concurrent requests
- Check whether your request ran
- Change parameters of the request's processing options
- Find the position of your request in the queues of available concurrent managers

For details on running the Requests concurrent program, see Section B.9, "Requests Concurrent Program" on page B-25.

A.6 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 Section B.9, "Requests Concurrent Program" on page B-25.

A.7 Checking BOM and Configuration Model Similarity

See Section 7.2, "Synchronizing BOM Data" on page 7-2 for more information on checking the similarity between the configuration model and the original BOM.

B

Concurrent Programs

This appendix explains how to use the Oracle Configurator concurrent programs that are available to the Configurator Developer and Configurator Administrator responsibility in Oracle Applications:

- Configurator Administration Concurrent Programs
- Server Administration Concurrent Programs
- Configuration Model Publication Concurrent Programs
- Populate and Refresh Configuration Models Concurrent Programs
- Model Synchronization Concurrent Programs
- Execute Populators in Model Concurrent Program
- Migration Concurrent Programs
- Publication Synchronization Concurrent Programs
- Requests Concurrent Program
  - Importing Data into Specific Tables
  - Show Tables to be Imported

For general information about running Oracle Configurator concurrent programs, see Section A.1 on page A-1.

B.1 Configurator Administration Concurrent Programs

The configurator administration concurrent programs are:

- View Configurator Parameters
- Modify Configurator Parameters

- Purge Configurator Tables

Configurator parameters are stored in the CZ_DB_SETTINGS table. See Section 4.4, "CZ_DB_SETTINGS Table" on page 4-7 for details about the parameters in that table.

B.1.1 View Configurator Parameters

The View Configurator Parameters concurrent program allows you to view the values of parameters in the CZ_DB_SETTINGS table.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Configurator Administration** > **View Configurator Parameters.**

Parameters

Table B–1 Parameters for the View 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 Section 4.4.3, "CZ_DB_SETTINGS Parameters" on page 4-8 for more information.
Setting ID	From the list of values, select the setting in the Oracle Configurator CZ_DB_SETTINGS Table where the CommitSize resides. See Section 4.4.3.7, "CommitSize" on page 4-13 for more information.

Output

The output containing the values for the specified SECTION_NAME and SETTING_ID are recorded in a log file (see Section B.9, "Requests Concurrent Program" on page B-25).

B.1.2 Modify Configurator Parameters

The Modify Configurator Parameters concurrent program allows you to change the values of parameters in the CZ_DB_SETTINGS table.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Configurator Administration > Modify Configurator Parameters**.

Parameters

Table B–2 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 Section 4.4.3, "CZ_DB_SETTINGS Parameters" on page 4-8.
Setting ID	From the list of values, select the setting in the Oracle Configurator CZ_DB_SETTINGS table you want to modify. See Section 4.4.3.7, "CommitSize" for more information on 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.
Value	Enter the value for the particular parameter. See Section 4.4 for more information on valid values for each of the settings in each SECTION_NAME.
Description	Enter a brief description for this value selection.

Output

The output containing the modified values of the CZ_DB_SETTINGS Parameters you specified are recorded in a log file (see Section B.9, "Requests Concurrent Program" on page B-25).

B.1.3 Purge Configurator Tables

The concurrent program Purge Configurator Tables physically removes all logically-deleted records in the tables and subschemas of the CZ schema. Periodically running this concurrent program improves database performance. See Chapter 8, "CZ Schema Maintenance" 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 Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Configurator Administration > Purge Configurator Tables**.

Parameters

There are no parameters required for this concurrent program.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.2 Server Administration Concurrent Programs

The server administration concurrent programs are:

- Define Remote Server
- Enable Remote Server
- View Servers
- Modify Server Definition

See Chapter 3, "Database Instances" for information about a multi-server environment requiring use of these concurrent programs.

B.2.1 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
- Database Instance publication applicability parameter drop-down list in Oracle Configurator Developer
- 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

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Server Administration > Define Remote Server**.

Parameters

Table B–3 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 drop-down list when creating a publication record and specifying the Database Instance applicability parameter. It is the name that is in the list of values when a Target or Source Instance parameter is needed for running a concurrent program, and is the name that appears in the list of values when enabling a remote server. For example, production.
Host Name	A TCP host name for this server where the Oracle Configurator schema is found. It can be an IP address or the actual name of the server. This is the actual machine. For example, myserver.

Table B–3 (Cont.) Parameters for the Define Remote Server Concurrent Program

Parameter	Description
DB Listener Port	A TCP port number on which this database server is listening for client connections. For example, 1523.
Instance Name	This 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	The 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 Section A.6, "Viewing Log Files" on page A-4.

B.2.2 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 loaded or 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 Y. For more information about importing data, see Chapter 5, "Populating the CZ Schema".

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Server Administration > Enable Remote Server**.

Parameters

Table B–4 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 Section A.6, "Viewing Log Files" on page A-4.

B.2.3 View Servers

The View Servers concurrent program writes each defined server's information to the concurrent program log.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Server Administration > View Servers**.

Parameters

There are no parameters required for this concurrent program.

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 Section A.6, "Viewing Log Files" on page A-4.

B.2.4 Modify Server Definition

The Modify Server Definition concurrent program allows you to change any 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 Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Server Administration > Modify Server**.

Parameters

Table B–5 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 drop-down list when creating a publication record and specifying the Database Instance applicability parameter. It is the name that is in the list of values when a Target or Source Instance parameter is needed for running a concurrent program, and is the name that appears in the list of values when enabling a remote server. For example, production.
Host Name	A TCP host name for this server where the Oracle Configurator schema is found. It can be an IP address or the actual name of the server. This is the actual machine. For example, myserver
DB Listener Port	A TCP port number on which this database server is listening for client connections.

Table B–5 (Cont.) Parameters for the Modify Server Definition Concurrent Program

Parameter	Description
Instance Name	This 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	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 Section A.6, "Viewing Log Files" on page A-4.

B.3 Configuration Model Publication Concurrent Programs

The publication concurrent programs include:

- Process Pending Publications
- Process a Single Publication

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 Publishing window.

These concurrent program 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 Section 17.2.3, "Publication Applicability Parameters" on page 17-8 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 on page B-4.

Running the publication concurrent programs includes BOM synchronization. For details, see Section 7.2.2, "Checking BOM and Model Similarity" on page 7-3 and Section 17.3, "Publishing a Configuration Model" on page 17-10.

B.3.1 Process Pending Publications

The Process Pending Publications concurrent program publishes all publications in the CZ_PB_MODEL_EXPORTS table that have a STATUS of PEN to their specified Database Instance applicability parameter.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database where the configuration model and its publication are defined.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Publish Configuration Models** > **Process Pending Publications**.

Parameters

There are no parameters required for this concurrent program.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.3.2 Process a Single Publication

The Process a Single Publication concurrent program publishes the selected publication to its specified Database Instance applicability parameter.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database where the configuration model and its publication are defined.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Publish Configuration Models** > **Process a Single Publication**.

Parameters

Table B–6 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 Publishing window in Oracle Configurator Developer, and 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 Section A.6, "Viewing Log Files" on page A-4.

B.4 Populate and Refresh Configuration Models Concurrent Programs

The concurrent programs for populating and refreshing configuration models are:

- Populate Configuration Models
- Refresh a Single Configuration Model
- Refresh All Imported Configuration Models
- Disable / Enable Refresh of a Configuration Model

Use the Populate/Refresh Configuration Models concurrent programs to import data into the CZ schema, including:

- Extracting BOM data into the correct format for transfer (Standard Import, only)
- Loading the data into the import tables (Standard Import, only)
- Populating the online CZ schema with the data from the import tables

For more information about data import, see Chapter 5, "Populating the CZ Schema".

Note: The Populate and Refresh Configuration Models concurrent programs do not provide an automated or scheduled mechanism that clears the import tables.

B.4.1 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 Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database into which you are importing data.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Populate/Refresh Configuration Models > Populate Configuration Models**.

Parameters

If no data is available in the list of values for the following parameter fields, see Section B.4.1.1, "Populate Configuration Models Concurrent Program Error Messages" on page B-13.

Table B–7 Parameters for the Populate Configuration Models Concurrent Program

Parameter	Description
Organization Code	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 Section A.6, "Viewing Log Files" on page A-4.

B.4.1.1 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:

```
'error 2019: connection description for remote database not found'
```

(The SQL statement that is currently running follows this message.)

Action:

1. The Configurator Administrator must run the Modify Server Definition concurrent program and Enable Remote Server for import (if one is not already selected). See Table B–5, " Parameters for the Modify Server Definition Concurrent Program" on page B-8 and Section B–4, " Parameters for the Enable Remote Server Concurrent Program" on page B-7.
2. Run Enable Remote Server if the enabled server is not LOCAL. See Table B–4, " Parameters for the Enable Remote Server Concurrent Program" on page B-7. Rerunning this concurrent program recreates the extraction views.

B.4.2 Refresh a Single Configuration Model

The Refresh a Single Configuration Model concurrent program updates the imported BOM data in the CZ schema when information in Oracle Applications Bills of Material and Inventory has changed.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database in which you are refreshing data.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Populate/Refresh Configuration Models > Refresh a Single Configuration Model**.

Parameters

Table B–8 lists the parameters used for both the Refresh a Single Configuration Model and Disable / Enable Refresh of a Configuration Model concurrent programs.

Table B–8 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, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.4.3 Refresh All Imported Configuration Models

The Refresh All Imported Configuration Models concurrent program updates all of your imported BOM data.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database in which you are refreshing data.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Populate/Refresh Configuration Models > Refresh All Imported Configuration Models**.

Parameters

There are no parameters required for this concurrent program.

B.4.4 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.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database containing the configuration model whose refresh is being controlled.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Populate/Refresh Configuration Models > Disable/Enable Refresh of a Configuration Model**.

Parameters

See Table B–8 on page B-14.

B.5 Model Synchronization Concurrent Programs

The model synchronization concurrent programs include:

- Check Model / Bill Similarity
- Check All Models / Bills Similarity
- Synchronize All Models

Check Model / Bill Similarity and Check All Models / Bills Similarity 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 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 can be run.

See Chapter 7, "Synchronizing Data"for more information.

B.5.1 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 of discrepancies, if any. See Section 7.2.3, "Criteria for BOM Similarity" on page 7-3 for information about the validation criteria used by this concurrent program.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database containing the configuration model that needs to be checked.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Model Synchronization > Check model/bill similarity**.

Parameters

Table B–9 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 concurrent program. Select the instance that contains the source BOM Model with which the configuration model needs to 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 Section B.5.4, "Model/Bill Similarity Check Report".

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.5.2 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 of discrepancies, if any. See Section 7.2.3, "Criteria for BOM Similarity" on page 7-3 for information about the validation criteria used by this concurrent program.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database containing the configuration models that need to be checked.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Model Synchronization > Check All models/bills similarity**.

Parameters

Table B–10 Check All Models/Bills Similarity Parameters

Parameter	Description
Target Instance	A list of available instances, as defined by the Define Remote Server concurrent program. Select the instance that contains the source BOM Model with which the configuration model needs to be synchronized.

Output

A report is generated with the results. See Section B.5.4, "Model/Bill Similarity Check Report".

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.5.3 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 concurrent program have been corrected (see Section B.5.4, "Model/Bill Similarity Check Report" on page B-19).

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database containing the configuration model that needs to be synchronized.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Model Synchronization > Synchronize All Models**.

Parameters

There are no parameters required for this concurrent program.

Output

A report is generated with the results. See Section B.5.4, "Model/Bill Similarity Check Report".

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.5.4 Model/Bill Similarity Check Report

The Model/Bill Similarity Check Report is generated every time you run the Check Model/Bill Similarity, Check All Models/Bills Similarity and Synchronize All Models 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 Section 7.2.3, "Criteria for BOM Similarity" on 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:

```
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.

```
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.
```

B.6 Execute Populators in Model Concurrent Program

The Execute Populators in Model concurrent Program is the same procedure as if you repopulated a Model in Oracle Configurator Developer by choosing **Tools > Repopulate** from the Model window. It is advantageous to run the Execute

Populators in Model concurrent program, as repopulating a Model in Oracle Configurator Developer is time consuming and the concurrent program can be scheduled to run at a specific time. For information about Populators, see *Oracle Configurator Developer User's Guide*.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the database containing the configuration model in which Populators should be executed.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Execute Populators in Model**.

Parameters

Table B–11 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 executed.
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, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.7 Migration Concurrent Programs

The migration concurrent programs are:

- Setup Configurator Data Migration
- Migrate Configurator Data

The migration concurrent programs move data from a source CZ schema to an empty target CZ schema. See Section 6.3, "Migrating Data from Another Oracle Configurator Schema" on page 6-8 for prerequisites before running the migration concurrent programs.

The source database server must be defined and enabled as the remote server (see Section B.2, "Server Administration Concurrent Programs" on page B-4).

B.7.1 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 Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the target database into which you are migrating data.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Migration > Setup Configurator Data Migration**.

Parameters

Table B–12 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 concurrent program.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file (see Section B.9, "Requests Concurrent Program" on page B-25).

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 table
- Schema versions for the source and target databases are not the same
- Difference in table structure

If any issues are found, correct them and run Setup Configuration Data Migration again.

B.7.2 Migrate Configurator Data

The Migrate Configurator Data concurrent program migrates the data from the source database instance to the target database instance.

Use the procedure described in Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program in the empty target database into which you are migrating data.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Migration > Migrate Configurator Data**.

Parameters

Table B–13 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 Section A.6, "Viewing Log Files" on page A-4.

B.8 Publication Synchronization Concurrent Programs

The publication synchronization concurrent programs are:

- Synchronize Cloned Target Data

- Synchronize Cloned Source Data

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 in order to establish the database link. See Section B.2, "Server Administration Concurrent Programs" on page B-4 for information about defining and enabling a remote server.

B.8.1 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 Section 7.3.1, "Synchronizing Publication Data after a Database Instance is Cloned" on page 7-6.

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 Section A.1, "Running Configurator Concurrent Programs" on page A-1 to run this concurrent program.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Publication Synchronization > Sychronize Cloned Target Data**.

Parameters

Table B–14 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 concurrent program.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-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 Publishing window in Oracle Configurator Developer displays Error in the **Status** column (see the *Oracle Configurator Developer User's Guide*).

B.8.2 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 Section 7.3.2.5, "Example of Synchronizing Publication Data on a Cloned Source" on page 7-11.

Before running the concurrent program, the cloned source database instance must be defined and enabled in order to establish the database link. See "Define Remote

Server" on page B-5 and "Enable Remote Server" on page B-6. This concurrent program is run from the cloned source database instance.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Publication Synchronization > Sychronize Cloned Source Data**.

Parameters

Table B–15 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 Section A.6, "Viewing Log Files" on page A-4.

B.9 Requests Concurrent Program

The Requests concurrent program enables you to:

- View all submitted concurrent program requests
- Determine whether your concurrent program has run
- Change parameters of the Request processing options
- Diagnose errors
- Find the position of your request in the queues of the available concurrent programs
- View the log file for a submitted concurrent program

- Submit a New Request

For additional information about the Request concurrent program, see the *Oracle Application User's Guide*.

Responsibility

Configurator Administrator or Configurator Developer

Navigation

Navigator window > **Requests**

Parameters

The Find Requests window presents five categories of Requests:

- My Completed Requests
- My Requests in Progress
- All My Requests
- Specific Requests

The requests display with the Request ID, Name, Parent, Phase, Status, Requestor, and Priority.

If your system administrator set the profile option Concurrent: Report Access Level to User, then the Requests window displays the concurrent requests for the current user.

If this profile option is set to Responsibility, then the Requests window displays all concurrent requests for the current responsibility in addition to the current user's requests.

- Submit a New Request

  All concurrent programs that are available in the Configurator Administrator window are also available in the Submit a New Request window. See Section B.9.1, "Importing Data into Specific Tables" on page B-27 for the navigation path. The parameter window for the selected Name is the same as if you ran the concurrent program from the Configurator Administrator.

Action

Determine the type of Request and click **Find**.

Output

The Request window displays the Request ID, Concurrent Program Name, Phase, Status, and Parameters that were entered when the concurrent program was chosen.

Selecting a particular Request and then clicking on the **View Log...** button opens a browser window displaying any errors or warnings for the selected concurrent program.

B.9.1 Importing Data into Specific Tables

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

Configurator Administrator

Navigation

Navigator window > **Requests**

In the Find Requests window, click **Submit a New Request**

In the Submit a New Request window accept the default and click **OK** to submit a single request.

Parameters

Table B–16 Import Data into Specific Tables

Parameter	Description
Name	This is a list of concurrent programs. Select the Select Tables To Be Imported from the drop-down 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 or Import

Table B–16 (Cont.) Import Data into Specific Tables

Parameter	Description
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.

Example B–1 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 the **Submit** button.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

B.9.2 Show Tables to be Imported

You can display the tables that are currently enabled for import.

Responsibility

Configurator Administrator

Navigation

Navigator window > **Requests**

In the Find Requests window, click **Submit a New Request**

In the Submit a New Request window accept the default and click **OK** to submit a single request.

Parameters

Table B–17 Show Tables to be Imported

Parameter	Description
Table Name	Enter the table in the CZ schema that you are querying the import disability.
Phase Name:	Enter either Export or Import for which you want to display the Import Enable setting.

Example B–2 Show Tables to be Imported

The following example displays the current Disable setting for the CZ_XFR_ TABLES.

```
Table Name: CZ_XFR_TABLES
Phase Name: Import
```

Action

After specifying the parameters click **OK**. In the Submit Request window, click the **Submit** button.

Output

To see if there are any errors or warnings for the concurrent program, you must examine the log file. See Section A.6, "Viewing Log Files" on page A-4.

C

CZ Subschemas

C.1 Oracle Configurator Subschemas

The following sections list the tables in each subschema. For detailed information about these and other tables, see the Configurator eTRM on Metalink, Oracle's technical support Web site.

C.1.1 ADMN Administrative Tables

CZ_DB_LOGS
CZ_DB_SETTINGS
CZ_DB_SIZES

C.1.2 CNFG Configuration Tables

CZ_ATP_REQUESTS
CZ_CONFIG_ATTRIBUTES
CZ_CONFIG_CONTENTS_V
CZ_CONFIG_DETAILS_V
CZ_CONFIG_HDRS
CZ_CONFIG_INPUTS
CZ_CONFIG_ITEMS
CZ_CONFIG_MESSAGES
CZ_CONFIG_EXT_ATTRIBUTES
CZ_PRICING_STRUCTURES
CZ_TERMINATE_MSGS
CZ_TERMINATE_MSGS_V

C.1.3 ITEM Item-Master Tables

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

C.1.4 LCE Logic for Configuration Tables

CZ_LCE_HEADERS
CZ_LCE_TEXTS

C.1.5 PB Publication Tables

CZ_EFFECTIVITY_SETS
CZ_EXT_APPLICATIONS_V
CZ_MODEL_APPLICABILITIES_V
CZ_MODEL_PUBLICATIONS
CZ_MODEL_USAGES
CZ_PB_CLIENT_APPS
CZ_PB_LANGUAGES
CZ_PB_MODEL_EXPORTS
CZ_PUBLICATION_USAGES

C.1.6 PROJ Project Structure Tables

CZ_DEVL_PROJECT
CZ_IMP_DEVL_PROJECTS
CZ_IMP_MODEL_REF_EXPLS
CZ_IMP_PS_NODES
CZ_MODEL_REF_EXPLS
CZ_POPULATORS
CZ_PS_NODES
CZ_PS_PROP_VALS

C.1.7 RULE Rule Tables

CZ_COMBO_FEATURES
CZ_DES_CHART_CELLS
CZ_DES_CHART_COLUMNS
CZ_DES_CHART_FEATURES
CZ_EXPRESSIONS
CZ_EXPRESSION_NODES
CZ_FILTER_SETS
CZ_FUNC_COMP_SPECS
CZ_GRID_CELLS
CZ_GRID_COLS
CZ_GRID_DEFS
CZ_RULES
CZ_RULE_FOLDERS

C.1.8 RP Repository Tables

CZ_REPOS_TREE_V
CZ_RP_DIRECTORY_V
CZ_RP_ENTRIES
CZ_SERVERS

C.1.9 UI User Interface Tables

CZ_IMP_INTL_TEXT
CZ_IMP_LOCALIZED_TEXTS
CZ_INTL_TEXTS
CZ_LOCALIZED_TEXTS
CZ_LOCALIZED_TEXTS_VL
CZ_UI_DEFS
CZ_UI_NODES
CZ_UI_NODE_PROPS
CZ_UI_PROPERTIES

C.1.10 XFR Transfer Specifications and Control Tables

CZ_XFR_FIELDS
CZ_XFR_PROJECT_BILLS
CZ_XFR_RUN_INFOS
CZ_XFR_RUN_RESULTS
CZ_XFR_STATUS_CODES

CZ_XFR_TABLES

D

Code Examples

This chapter contains code examples that support other chapters of this document. These examples are fuller and longer than the examples provided in the rest of this document, which are often fragments. See the cited background sections for details.

Table D–1 Code Examples Provided

Purpose of Example	Example
Section D.1, "Pricing and ATP Callback Procedures"	Example D–1, "Example of Single-item Callback Pricing Procedure" Example D–2, "Example of Multiple-item Callback Pricing Procedure" Example D–3, "Example of Callback ATP Procedure"
Section D.2, "Implementing a Return URL Servlet"	Example D–4, "Example Return URL Servlet (Checkout.java)"

You should consult these other documents for details on the tasks described in this section:

- For information on how to write and compile Functional Companions, and on how to incorporate them into your configuration model, see the *Oracle Configuration Interface Object (CIO) Developer's Guide.*
- For information on how to install Functional Companions, see the *Oracle Configurator Installation Guide.*
- For an explanation of updating configurations, see the *Oracle Configurator Developer User's Guide.*
- For an details on how to build a configuration model that enables you to update configurations, see the *Oracle Configurator Developer User's Guide.*

D.1 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:

- Chapter 13, "Pricing and ATP in Oracle Configurator" on page 13-1
- Section 13.3.2, "The Pricing Callback Interface" on page 13-6
- Section 13.3.3, "The ATP Callback Interface" on page 13-11
- Section 9.5.5, "Pricing Parameters" on page 9-13
- Section 9.5.6, "ATP Parameters" on page 9-14

Example D–1 Example of Single-item Callback Pricing Procedure

```
PROCEDURE price_single_item (configurator_session_key IN VARCHAR2,
                             price_type IN VARCHAR2,
                             ps_node_id IN NUMBER,
                             item_key IN VARCHAR2,
                             quantity IN NUMBER,
                             uom_code IN VARCHAR2,
                             list_price OUT NUMBER,
                             selling_price OUT NUMBER,
                             msg_data OUT VARCHAR2) AS
BEGIN

  IF item_key IS NULL THEN
  -- hard-coded price amounts
    list_price := 2.0;
    selling_price := 2.0;
  ELSE
    list_price := 1.0;
    selling_price := 1.0;
  END IF;

  -- debugging information
  msg_data := configurator_session_key || ' : ' || price_type || ' : ' ||
              item_key || ' : ' || uom_code;

END price_single_item;
```

Example D–2 Example of Multiple-item Callback Pricing Procedure

```
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 D–3 Example of Callback ATP Procedure

```
  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;
```

D.2 Implementing a Return URL Servlet

Example D–4 is 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 an HTML frame that takes the place of the Oracle Configurator window after your user has closed the window and saved the results of the configuration session.

See the following sections for background:

- Section 10.8, "The Return URL" on page 10-10
- Section 10, "Session Termination" on page 10-1
- Section 10.5, "Submission" on page 10-3
- Section 10.5.1, "Configuration Status" on page 10-4
- Section 10.5.2, "Configuration Outputs" on page 10-6

The parts of the code that you should customize to work with a different configuration output element than `valid_configuration` are typographically emphasized.

Note the use of `top.location` in the example to cause the servlet output to replace the contents of the runtime Oracle Configurator window.

Example D–4 Example Return URL Servlet (Checkout.java)

```
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 = \"/configurator/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 of Terms and Acronyms

This glossary contains definitions that you may need while working with Oracle Configurator.

Active Model

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 Active Model must be generated either in **Oracle Configurator Developer** or programmatically in order to access the configuration model at **runtime**.

API

Application Programming Interface

applet

A Java application running inside a Web browser. *See also* **Java** and **servlet**.

application architecture

The software structure of an application at **runtime**. Architecture affects how an application is used, maintained, extended, and changed.

architecture

See **application architecture**.

ATO

Assemble to Order

ATP

Available to Promise

attribute

The defining characteristic of something. Models have attributes such as Effectivity Sets and Usage. Components, Features, and Options have attributes such as Name, Description, and Effectivity.

benchmark

Represents performance data collected during **runtime** tests under various conditions that emulate expected and extreme use of the product.

beta

An external release, delivered as an installable application, and subject to acceptance, **validation**, and **integration testing**. Specially selected and prepared **end users** may participate in beta testing.

bill of material

A list of Items associated with a parent Item, such as an assembly, and information about how each Item relates to that parent Item.

Bills of Material

The application in Oracle Applications in which you define a **bill of material**.

BOM

See **bill of material**.

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** 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 **option**s: true or false.

bug

See **defect**.

build

A specific **instance** of an application during its construction. A build must have an install program early in the project so that application **implementer**s can **unit test** their latest work in the context of the entire available application.

CIO

See **Oracle Configuration Interface Object (CIO)**.

client

A **runtime** program using a **server** to access functionality shared with other clients.

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 **Feature**s, **Total**s, **Resource**s, **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 **Feature**s in the Model to control the allowable combinations of **Option**s. *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 **Option**s 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 **Feature**s, that is configurable and instantiable. An **Oracle Configurator Developer** node type that represents a configurable element of a **Model**. Corresponds to one UI screen of selections in a runtime **Oracle Configurator**.

Component Set

An element of the **Model** that contains a number of instantiated **Component**s of the same type, where each Component of the set is independently configured.

concurrent manager

A process manager that coordinates the **concurrent process**es generated by **user**s' **concurrent request**s. 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 program

Executable code (usually written in SQL*Plus or Pro*C) that performs the function(s) of a requested task. Concurrent programs are stored procedures that

perform actions such as generating reports and copying data to and from a database.

concurrent request

A user-initiated request issued to the concurrent processing facility to submit a non-interactive task, such as running a report.

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 Interface Object

See **Oracle Configuration Interface Object (CIO)**.

configuration model

Represents all possible configurations of the available **option**s, and consists of **model structure** and **rules**. It also commonly includes **User Interface** definitions and **Functional Companion**s. A configuration model is usually accessed in a **runtime Oracle Configurator window**. *See also* **model**.

configuration rules

The **Oracle Configurator Developer Logic rules**, **Compatibility rule**s, **Comparison rule**s, **Numeric rule**s, and **Design Chart**s available for defining **configuration**s. *See also* **rules**.

configuration session

The time from launching or invoking to exiting **Oracle Configurator**, during which **end user**s 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 hosting application so **end user**s

can make selections resulting in valid **configuration**s. *Compare* **Oracle Configurator**.

connectivity

The connection between **client** and database **server** that allows data communication.

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**.

Container Model

A type of **BOM Model** that you import from Oracle **Bills of Material** into **Oracle Configurator Developer** to create configuration models containing **connectivity** and trackable components. Configurations created from Container Models can be tracked and updated in Oracle Install Base

context

The surrounding text or conditions of something.

Determines which context-sensitive segments of a flexfield in the Oracle Applications database are available to an application or **user**. Used in defining **configuration attribute**s.

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 decrementing a total value, such as specifying the quantity of a **Resource** used.

count

The number or quantity of something, such as selected **option**s. *Compare* **instance**.

CRM

See **Customer Relationship Management**

CTO

Configure to Order

customer

The person for whom products are configured by **end users** of the **Oracle Configurator** or other **ERP** and **CRM** applications. Also the end users themselves directly accessing **Oracle Configurator** in a Web store or kiosk.

customer-centric extensions

See **customer-centric views**.

customer-centric views

Optional extensions to core functionality that supplement configuration activities with **rules** for **preselection**, **validation**, and **intelligent views**. View capabilities include generative geometry, drawings, sketches and schematics, charts, performance analyses, and **ROI** calculations.

Customer Relationship Management

The aspect of the enterprise that involves contact with customers, from lead generation to support services.

customer requirements

The needs of the customer that serve as the basis for determining the configuration of products, **systems**, and services. Also called needs assessment. *See* **guided buying or selling**.

CZ

The product shortname for **Oracle Configurator** in Oracle Applications.

data import

Populating the **Oracle Configurator schema** with enterprise data from **ERP** or legacy systems via **import tables**.

Data Integration Object

Also known as the DIO, the Data Integration Object is a **server** in the **runtime** application that creates and manages the interface between the **client** (usually a **user interface**) and the **Oracle Configurator schema**.

data maintenance environment

The environment in which the runtime **Oracle Configurator** data is maintained.

data source

A programmatic reference to a database. Referred to by a data source name (DSN).

DBMS

Database Management System

default

A predefined value. In a **configuration**, the automatic selection of an **option** based on the **preselection** rules or the selection of another option.

Defaults rule

An **Oracle Configurator Developer** Logic rule that determines the logic state of **Feature**s or **Option**s 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).

defect

A failure in a product to satisfy the **user**s' requirements. Defects are prioritized as critical, major, or minor, and fixes range from corrections or workarounds to enhancements. Also known as a bug.

defect tracking

A system of identifying defects for managing additional tests, testing, and approval for release to **user**s.

deliverable

A work product that is specified for review and delivery.

demonstration

A presentation of the tested application, showing a particular usage scenario.

Design Chart

An **Oracle Configurator Developer** rule type for defining advanced Explicit Compatibilities interactively in a table view.

design review

A technical review that focuses on application or **system** design.

developer

The person who uses **Oracle Configurator Developer** to create a **configurator**. *See also* **implementer** and **user**.

Developer

The tool (**Oracle Configurator Developer**) used to create **configuration models**.

DHTML

Dynamic Hypertext Markup Language

DIO

See **Data Integration Object**.

distributed computing

Running various **components** of a **system** on separate machines in one network, such as the database on a database **server** machine and the application software on a Web server machine.

DLL

Dynamically Linked Library

DSN

See **data source**.

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**.

enterprise

The **systems** and **resources** of a business.

environment

The arena in which software tools are used, such as operating system, applications, and **server** processes.

ERP

Enterprise Resource Planning. A software system and process that provides automation for the customer's back-room operations, including order processing.

Excludes rule

An **Oracle Configurator Developer** Logic rule determines the logic state of **Feature**s or **Option**s 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 rule**.

extended functionality

A release after delivery of core functionality that extends that core functionality with **customer-centric views**, more complex proposal generation, discounting, quoting, and expanded integration with **ERP**, **CRM**, and third-party software.

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 **Option**s.

Functional Companion

An extension to the **configuration model** beyond what can be implemented in Configurator Developer.

An object associated with a **Component** that supplies methods that can be used to initialize, validate, and generate **customer-centric views** and outputs for the **configuration**.

functional specification

Document describing the functionality of the application based on **user** requirements.

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 **option**s and exclude others based on the **end user's** responses.

host application

An application within which **Oracle Configurator** is embedded as integrated functionality, such as Order Management or *i*Store.

HTML

Hypertext Markup Language

ICX

Inter-Cartridge Exchange

implementation

The stage in a project between defining the problem by selecting a configuration technology vendor, such as Oracle, and deploying the completed configuration application. The implementation stage includes gathering requirements, defining test cases, designing the application, constructing and testing the application, and delivering it to **end user**s. *See also* **developer** and **user**.

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 rule

An **Oracle Configurator Developer** Logic rule that determines the logic state of **Feature**s or **Option**s 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 rule**.

import server

A database **instance** that serves as a source of data for **Oracle Configurator**'s Populate, Refresh, and Synchronization concurrent processes. The import server is sometimes referred to as the remote server.

import tables

Tables mirroring the Oracle Configurator schema Item Master structure, but without integrity constraints. Import tables allow batch population of the Oracle Configurator schema's Item Master. Import tables also store extractions from Oracle Applications or **legacy data** that create, update, or delete records in the Oracle Configurator schema **Item Master**.

incremental construction

The process of organizing the construction of the application into **build**s, where each build is designed to meet a specified portion of the overall requirements and is **unit test**ed.

initialization message

The **XML** message sent from a **host application** to the **Oracle Configurator Servlet**, containing data needed to initialize the runtime Oracle Configurator. *See also* **termination message**.

install program

Software that sets up the local machine and installs the application for testing and use.

Instance

An **Oracle Configurator Developer** attribute of a **component's node** that specifies a minimum and maximum value. *See also* **instance**.

instance

A **runtime** occurrence of a **component** in a configuration. *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**.

integration

The process of combining multiple software **component**s and making them work together.

integration testing

Testing the interaction among software programs that have been integrated into an application or **system**. *Compare* **unit test**.

intelligent views

Configuration output, such as reports, graphs, schematics, and diagrams, that help to illustrate the value proposition of what is being sold.

IS

Information Services

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 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**.

Java

An object-oriented programming language commonly used in internet applications, where Java applications run inside Web browsers and **server**s. *See also* **applet** and **servlet**.

LAN

Local Area Network

LCE

Logical Configuration Engine. *Compare* **Active Model**.

legacy data

Data that cannot be imported without creating custom extraction programs.

load

Storing the **configuration model** data in the **Oracle Configurator Servlet** on the Web server. Also, the time it takes to initialize and display a configuration model if it is not preloaded.

The burden of transactions on a **system**, commonly caused by the ratio of **user** connections to CPUs or available memory.

log file

A file containing errors, warnings, and other information that is output by the running application.

Logic rules

Logic rules directly or indirectly set the logical state (User or Logic True, User or Logic False, or **Unknown**) of **Feature**s and **Option**s 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 rule**, **Requires rule**, **Excludes rule**, and **Negates rule**.

maintainability

The characteristic of a product or process to allow straightforward **maintenance**, alteration, and extension. Maintainability must be built into the product or process from inception.

maintenance

The effort of keeping a **system** running once it has been deployed, through **defect** fixes, procedure changes, infrastructure adjustments, data replication schedules, and so on.

Metalink

Oracle's technical support Web site at:

`http://www.oracle.com/support/metalink/`

Model

The entire hierarchical "tree" view of all the data required for **configuration**s, including **model structure**, variables such as **Resource**s and **Total**s, 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

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-driven UI

The graphical views of the **model structure** and rules generated by **Oracle Configurator Developer** to present **end users** with interactive product selection based on **configuration models**.

model structure

Hierarchical "tree" view of data composed of **elements** (**Models**, **Components**, **Features**, **Options**, **BOM Models**, **BOM Option Class nodees**, **BOM Standard Item nodes**, **Resources**, and **Totals**). May include reusable **components** (**References**).

MS

Microsoft Corporation

Negates rule

A type of **Oracle Configurator Developer** Logic rule 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). *See* **Excludes rule**.

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 node**.

Numeric rule

An **Oracle Configurator Developer** rule type that express constraint among model elements in terms of numeric relationships. *See also*, **Contributes to** and **Consumes from**.

OC

See **Oracle Configurator**.

ODBC

Open Database Connectivity. A database access method that uses drivers to translate an application's data queries into **DBMS** commands.

OCD

See **Oracle Configurator Developer**.

opportunity

The workspace in Oracle Sales Online in which products, **systems**, and services are configured, quotes and proposals are generated, and orders are submitted.

option

A logical selection made by the **end user** when configuring a **component**.

Option

An element of the **Model**. A choice for the value of an enumerated **Feature**.

Oracle Configuration Interface Object (CIO)

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 **Active Model**.

The CIO is the **API** that supports creating and navigating the Model, querying and modifying selection states, and saving and restoring **configurations**.

Oracle Configurator

The product consisting of development tools and **runtime** applications such as the **Oracle Configurator schema**, **Oracle Configurator Developer**, and runtime Oracle Configurator. Also the runtime Oracle Configurator variously packaged for use in networked or Web deployments.

Oracle Configurator architecture

The three-tier **runtime** architecture consists of the **User Interface**, the **Active Model**, and the **Oracle Configurator schema**. The application development architecture consists of **Oracle Configurator Developer** and the Oracle Configurator schema, with test instances of a runtime **Oracle Configurator**.

Oracle Configurator Developer

The suite of tools in the **Oracle Configurator** product for constructing and maintaining **configurator**s.

Oracle Configurator engine

Also **LCE**. *Compare* **Active Model**.

Oracle Configurator 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**.

Oracle Configurator Servlet

Vehicle for **Oracle Configurator** containing the UI Server.

Oracle Configurator window

The **user interface** that is launched by accessing a **configuration model** and used by **end user**s to make the selections of a **configuration**.

Oracle SellingPoint Application

No longer available or supported.

output

The output generated by a **configurator**, such as quotes, proposals, and **customer-centric views**.

performance

The operation of a product, measured in throughput and other data.

Populator

An entity in **Oracle Configurator Developer** that creates **Component**, **Feature**, and **Option node**s from information in the **Item Master**.

preselection

The default state in a **configurator** that defines an initial selection of **Component**s, **Feature**s, and **Option**s for configuration.

A process that is implemented to select the initial element(s) of the **configuration**.

product

Whatever is ordered and delivered to customers, such as the output of having configured something based on a model. Products include intangible entities such as services or contracts.

project manager

A member of the project team who is responsible for directing the project during implementation.

project plan

A document that outlines the logistics of successfully implementing the project, including the schedule.

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

A kind of compatibility relationship where the allowable combinations of **Option**s are specified implicitly by relationships among Property values of the Options.

prototype

A construction technique in which a preliminary version of the application, or part of the application, is built to facilitate **user** feedback, prove feasibility, or examine other implementation issues.

PTO

Pick to Order

publication

A unique deployment of a **configuration model** (and optionally a **user interface**) that enables a developer to control its availability from hosting 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.

QA

Quality Assurance

RAD

Rapid Application Development

RDBMS

Relational Database Management System

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**.

regression test

An automated test that ensures the newest **build** still meets previously tested requirements and functionality. *See also* **incremental construction**.

Requires rule

An **Oracle Configurator Developer** Logic rule that determines the logic state of **Feature**s or **Option**s 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 rule**.

resource

Staff or equipment available or needed within an enterprise.

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**.

reusable component

See **reference** and **model structure**.

reusability

The extent to and ease with which parts of a **system** can be put to use in other systems.

RFQ

Request for Quote

ROI

Return on Investment

rules

Also called business rules or **configuration rules**. 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 rules** and **Numeric rule**.

runtime

The environment and context in which applications are run, tested, or used, rather than developed.

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**.

sales configuration

A part of the sales process to which configuration technology has been applied in order to increase sales effectiveness and decrease order errors. Commonly identifies **customer requirements** and product configuration.

schema

The tables and objects of a data model that serve a particular product or business process. *See* **Oracle Configurator schema**.

SCM

Supply Chain Management

server

Centrally located software processes or hardware, shared by **client**s.

servlet

A Java application running inside a Web server. *See also* **Java**, **applet**, and **Oracle Configurator Servlet**.

SFA

Sales Force Automation

solution

The deployed **system** as a response to a problem or problems.

SQA

Software Quality Assurance

SQL

Structured Query Language

system

The hardware and software **component**s and infrastructure integrated to satisfy functional and **performance** requirements.

termination message

The **XML** message sent from the **Oracle Configurator Servlet** to a **host application** after a **configuration session**, containing configuration outputs. *See also* **initialization message**.

test case

A description of inputs, execution instructions, and expected results that are created to determine whether a specific software feature works correctly or a specific requirement has been met.

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**.

UI

See **User Interface**.

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**.

unit test

Execution of individual routines and modules by the application **implementer** or by an independent test consultant to find and resolve **defect**s in the application. *Compare* **integration testing**.

update

Moving to a new version of something, independent of software release. For instance, moving a production **configurator** to a new version of a **configuration model**, or changing a **configuration** independent of a model **update**.

upgrade

Moving to a new release of **Oracle Configurator** or **Oracle Configurator Developer**.

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 part of **Oracle Configurator architecture runtime** architecture that is generated from the **model structure** and provides the graphical views necessary to create **configuration**s interactively. Interacts with the **Active Model** and data to give **end user**s access to customer requirements gathering, product selection, and **customer-centric views**.

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**.

user requirements

A description of what the **configurator** is expected to do from the **end user's** perspective.

user's guide

Documentation on using the application or **configurator** to solve the intended problem.

validation

Tests that ensure that configured **component**s will meet specific criteria set by an enterprise, such as that the components can be ordered or manufactured.

Validation

A type of **Functional Companion** that is implemented to ensure that the configured **component**s will meet specific criteria.

VAR

Value-Added Reseller

variable

Parts of the **Model** that are represented by **Total**s, **Resource**s, or numeric **Feature**s.

VB

Microsoft Visual Basic. Programming language in which portions of **Oracle Configurator Developer** are written.

verification

Tests that check whether the result agrees with the specification.

WAN

Wide Area Network

Web

The portion of the Internet that is the World Wide Web.

WIP

Work In Progress

XML

Extensible Markup Language, a highly flexible markup language for transferring data between **Web** applications. Used for the **initialization message** and **termination message** of the **Oracle Configurator Servlet**.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X