

Oracle® Configurator

Fusion Configurator Engine Guide

Release 12.1

Part No. E14325-04

August 2010

Oracle Configurator Fusion Configurator Engine Guide, Release 12.1

Part No. E14325-04

Copyright © 1999, 2010, Oracle and/or its affiliates. All rights reserved.

Primary Author: Mark Sawtelle

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third party content, products and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third party content, products or services.

Contents

Send Us Your Comments

Preface

1 Introduction to the Fusion Configurator Engine

Introduction	1-1
Key Features of the Fusion Configurator Engine.....	1-2
Rules and the Fusion Configurator Engine.....	1-2
Managing Component Instances at Runtime.....	1-3
Auto-Complete Configuration.....	1-3
Additional Features of the Fusion Configurator Engine.....	1-5

2 Preparing to Use the Fusion Configurator Engine

Upgrading Oracle Configurator to use the Fusion Configurator Engine	2-1
Configurator Preferences Page.....	2-1
Profile Options.....	2-1
Converting Existing Models to Use the Fusion Configurator Engine.....	2-7

3 Building a Configuration Model Using the Fusion Configurator Engine

Model Structure.....	3-1
General Area of the Workbench.....	3-2
Model References.....	3-3
Properties.....	3-3
Domain Ordering Setting.....	3-16
Require End-User Input Setting.....	3-19
Text Features.....	3-19

Counted Option Features.....	3-20
Totals and Resources.....	3-21
Initial Values.....	3-21
BOM Nodes.....	3-22
BOM Model References.....	3-23
Connectors.....	3-24
Effectivity.....	3-26
Configuration Rules.....	3-27
Logic States.....	3-27
Rule Classes.....	3-27
Compatibility Rules.....	3-32
Accumulator Rules.....	3-32
Configurator Extensions.....	3-34
Fusion Configurator Engine and the Constraint Definition Language	3-37
Rule Import for Fusion Configurator Engine Models	3-45
Creating and Editing a User Interface.....	3-48
User Interface Master Templates.....	3-48
User Interface Content Templates	3-49
Control Templates.....	3-50
Utility Templates.....	3-52
Message Templates.....	3-54
Button Bar Templates.....	3-56
Other Templates.....	3-59
Changes to Existing User Interface Content Templates.....	3-60
Layout Regions.....	3-61
User Interface Elements.....	3-61
User Interface Actions.....	3-62
Displaying a Processing Page at Runtime.....	3-64
Runtime Icons and Images.....	3-66
User Interface Definition.....	3-67
Unit Testing a Configuration Model Using the Model Debugger.....	3-67

4 Runtime Behavior of the Fusion Configurator Engine

Domain Display and Availability	4-1
Logic State Display.....	4-1
Runtime Configurator Flows and Behavior.....	4-2
Auto-Complete Configuration.....	4-2
Instance Management.....	4-4
Finish Configuration.....	4-9
Conflict Handling and Resolution	4-9

Restoring a Completed Configuration.....	4-11
------------------------------------------	------

5 Configuration Attributes for Fusion Configurator Engine Models

About Configuration Attributes.....	5-1
Tasks for Adding Configuration Attributes to an FCE Model.....	5-5
Setting Up Descriptive Flexfields.....	5-6
Adding Attribute Features.....	5-8
Associating Attribute Features to Flexfield Segments.....	5-10
Associating BOM Nodes with Attribute Features.....	5-10
Defining the Configurator Extension Rule.....	5-12
Access to Configuration Attribute Data.....	5-14
Special Considerations.....	5-17
Maintaining the Configuration Attributes Setup.....	5-19
Using Configuration Attributes in the Downstream Application.....	5-19

6 CIO Emulation for the FCE

About CIO Emulation for the FCE.....	6-1
Why CIO Emulation for the FCE Is Needed.....	6-1
Intended Audience for CIO Emulation for the FCE.....	6-2
Elements of CIO Emulation for the FCE.....	6-2
Differences Between FCE and CIO Emulation.....	6-4
Candidate Implementations for Rewriting.....	6-8
Limitations of CIO Emulation for the FCE.....	6-9
Tasks for Implementing CIO Emulation for the FCE.....	6-10
Requirements for Implementing CIO Emulation.....	6-10
Converting Source Files with Substitution.....	6-11
What the Substitution Script Does.....	6-11
Running the Script.....	6-15
Syntax and Parameters.....	6-16
Custom Substitution.....	6-18
Output of the Script.....	6-20
Errors from the Script.....	6-22
Compiling and Archiving Converted Files.....	6-22
Converting Configurator Extension Rules.....	6-23
Verifying Post-Conversion Behavior.....	6-26

Common Glossary for Oracle Configurator

Index

Send Us Your Comments

Oracle Configurator Fusion Configurator Engine Guide, Release 12.1

Part No. E14325-04

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

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

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

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

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

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

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

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

Preface

Intended Audience

Welcome to Release 12.1 of the *Oracle Configurator Fusion Configurator Engine Guide*.

Important: This document describes only the Fusion Configurator Engine (FCE), which constitutes part of Oracle Configurator functionality. It is essential to read this document in conjunction with other documents in the Oracle Configurator documentation set, as described in this section.

This guide is intended for those who are in the process of planning, designing, building, or deploying configuration models using Oracle Configurator Developer, or have already deployed configuration models in a runtime Oracle Configurator. This document assumes you are experienced with both Oracle Configurator Developer and Oracle Configurator, have attended Oracle Configurator training classes available through Oracle University, and have read the *Oracle Configurator Developer User's Guide*.

The *Oracle Configurator Developer User's Guide* contains essential information about building and deploying configuration models using Oracle Configurator Developer. Much of the content in that guide is relevant whether you are building a configuration model for the Original Configurator Engine (OCE) or the Fusion Configurator Engine (FCE). However, all procedures, settings, modeling techniques, runtime behavior, and areas within the Configurator Developer user interface that are specific to the FCE are currently available only in this guide. Therefore, Oracle recommends that you use this guide as a supplement to the latest version of the *Oracle Configurator Developer User's Guide*.

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

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at <http://www.fcc.gov/cgb/consumerfacts/trs.html>, and a list of phone numbers is available at <http://www.fcc.gov/cgb/dro/trsphonebk.html>.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. 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. Accessibility standards will continue to evolve over time, and Oracle 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 more information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

1 Introduction to the Fusion Configurator Engine

This chapter describes at a high level the main features and benefits of the FCE. The remainder of this document describes the various settings that enable you to build FCE Models in Configurator Developer, how to convert existing Models to FCE Models, and the general behavior of FCE Models in a runtime Oracle Configurator.

2 Preparing to Use the Fusion Configurator Engine

3 Building a Configuration Model Using the Fusion Configurator Engine

4 Runtime Behavior of the Fusion Configurator Engine

5 Configuration Attributes for Fusion Configurator Engine Models

This chapter describes how to set up configuration attributes for Models that use the Fusion Configurator Engine.

6 CIO Emulation for the FCE

Common Glossary for Oracle Configurator

Related Information Sources

For a full list of documentation resources for Oracle Configurator, see the Oracle Configurator Release Notes for this release.

For a full list of documentation resources for Oracle Applications, see Oracle Applications Documentation Resources, on MetaLink, Oracle's technical support Web site.

Additionally, be sure you are familiar with current release or patch information for Oracle Configurator on MetaLink, Oracle's technical support Web site.

Integration Repository

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

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

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

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

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

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

When you use Oracle E-Business Suite to modify your data, Oracle E-Business Suite

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

Introduction to the Fusion Configurator Engine

This chapter describes at a high level the main features and benefits of the FCE. The remainder of this document describes the various settings that enable you to build FCE Models in Configurator Developer, how to convert existing Models to FCE Models, and the general behavior of FCE Models in a runtime Oracle Configurator.

This chapter covers the following topics:

- Introduction
- Key Features of the Fusion Configurator Engine

Introduction

The Fusion Configurator Engine (FCE) is written in the Java programming language and is based on Constraint Programming technology. In the constraint-based programming paradigm, relations between variables can be stated by defining *constraints*. Constraints specify the characteristics of a desirable solution to a configuration problem, rather than a step or sequence of steps that must be executed to create such a solution. After a user makes selections, the FCE automatically finds a solution that includes the user's selections and satisfies all of the constraints.

Compared to Models that use the Original Configurator Engine (OCE), FCE Models provide a greater degree of guidance in a runtime User Interface. This helps Oracle Configurator end users make better, more educated choices, resulting in fewer errors and allowing products to be configured more quickly.

FCE Models also enable end users to complete a configuration automatically using preferences that you define in Oracle Configurator Developer. These preferences consist of **Defaults** and **Search Decisions**, and are described in more detail in Rule Classes, page 3-27.

The benefits of using the FCE and leveraging its constraint-based technology include

reduced time spent developing configuration models, the ability to lead end users to the Modeler's preferred solution, and a product architecture that is easier to maintain.

Note: The Telecommunications Services Ordering (TSO) functionality is not available in the current release of the Fusion Configurator Engine. For information about TSO in previous releases, see the *Oracle Telecommunications Service Ordering Process Guide*.

Important: Connectors are not supported in the current release of the Fusion Configurator Engine. The information about Connectors provided in this document is only for planning purposes. However, Connectors continue to be supported for models using the Original Configurator Engine.

Key Features of the Fusion Configurator Engine

Rules and the Fusion Configurator Engine

This section describes functionality related to configuration rules that is available only when using the FCE.

When using the FCE, you can:

- Classify configuration rules as Constraints, Defaults (also called *soft constraints*), or Search Decisions.

For details, see Rule Classes, page 3-27.

- Specify the order in which Oracle Configurator evaluates rules that are defined as Defaults or Search Decisions.

For details, see Specifying a Sequence for Defaults and Search Decisions, page 3-31.

- Control the relative and absolute quantity of a BOM Model node by defining constraints in Configurator Developer.

- Define a rule that requires an end user to connect an optional Connector, or prevents a Connector from being connected.

For details, see the examples using the System Property `ConnectionCount` under Model Node System Properties, page 3-3 and Connectors and Configuration Rules, page 3-26.

- Define rules that control how many instances of a component can be created at runtime.

For details, see the examples using the System Property `InstanceCount` under

Model Node System Properties, page 3-3 and Instance Management, page 4-4.

- Define Accumulator Rules, which add or subtract a value from a variable at runtime, and which replace Numeric Rules.

For details, see Accumulator Rules, page 3-32.

- Define rules that specify default values for Numeric Features, Boolean Features, Totals, and Resources.

See Initial Values, page 3-21.

Managing Component Instances at Runtime

Important: See *Oracle Configurator Release Notes, Release 12.1.1* (On MetaLink, Oracle's technical support Web site) for background on important enhancements and changes to runtime instance management.

The following are true at runtime in FCE Models that support instantiation:

- Instances can be created automatically to satisfy a constraint. This is called *dynamic instantiation*.

For example, an instance may be created when no target instance exists for a required Connector, or when the minimum instances setting on an Instance Set is increased.

(In the OCE, instances can only be created by the end user or by a Configurator Extension (CX).)

- An end user can remove an instance from an Instance Set without deleting it from the configuration.
- An end user can add an existing instance to an Instance Set, rather than creating an entirely new instance. See The Unassigned Instance Pool, page 4-7 for background.
- An end user can copy an instance in an Instance Set and later add it to a set of the same type.

For details, and examples using the System Property `InstanceCount`, see Model Node System Properties, page 3-3 and Instance Management, page 4-4.

Auto-Complete Configuration

An important feature of the Fusion Configurator Engine is the ability to complete a configuration automatically. This process, which end users can invoke at runtime by

clicking the Finish button, is known as *Auto-Complete*. Auto-Complete is useful, for example, when end users care only about a subset of all available items and want to quickly create a valid and complete configuration. In this case, end users can manually provide the inputs that they care about, supply any inputs that are explicitly required by the Model definition, and then allow Auto-Complete to complete the configuration. After examining the result of Auto-Complete, the end user can save and exit, or make further changes before exiting. Details about how the process functions at runtime are explained in Auto-Complete Configuration, page 4-2.

When an Oracle Configurator end user invokes Auto-Complete, the FCE searches for a solution to the current configuration problem by providing inputs of the appropriate type for all unbound variables in the configuration and applying requirements and (optionally) preferences that you define for the Model in Configurator Developer.

- Requirements, which must be satisfied for a complete solution, are necessary for Auto-Complete to bind the variables. You define requirements by defining rules with an associated Rule Class, page 3-27 of Constraint.
- Preferences, which can be unsatisfied in a complete solution, are not necessary for Auto-Complete to bind the variables, but enable you to shape the final solution in a given direction. You define preferences by:
 - Defining rules with an associated Rule Class, page 3-27 of Default or Search Decision
 - Specifying a Sequence for Defaults and Search Decisions, page 3-31
 - Indicating how you want Auto-Complete to provide values for unbound Numeric and Boolean Features (see Domain Ordering Setting, page 3-16)
 - Specifying minimum and maximum values for all Model nodes (see Tip, below).

Tip: For optimal performance of the Auto-Complete process, Oracle strongly recommends narrowing the domains to be searched. To narrow the search domains, set maximums as low as possible and minimums as high as possible for the following node types: Numeric Features, Totals, and Resources (set the values); instantiable Components and Model References (set the number of instances); Option Features (set the number of selections); BOM components (set the quantities, in Oracle Bills of Material).

Variables

The term "variable" is used in this document to refer to any item that must be bound; that is, that requires selection or a value at runtime. For example, a BOM Option Class that requires at least one of its children to be selected, or a Numeric Feature that

requires a value. A variable is bound when it is selected or excluded (or a value is entered) by the end user, the propagation of a rule, or the Auto-Complete process. A configuration is not complete until *all* variables are bound.

Note: Text Features are the only type of variable that Auto-Complete cannot bind. For details, see Aspects of Auto-Complete Behavior, page 4-3.

Other examples of variables include:

- Boolean Feature: This type of node is bound at runtime when its domain is reduced to a single value (true or false).

Note: The term "domain" is explained in Domain Ordering Setting, page 3-16.

- Option Feature: This type of node is bound at runtime when its Selection Count is reduced to a single value that is equal to the number of selected options.

For example, OF1 has a Minimum and Maximum Selections of 1 and 4, respectively. At runtime, the valid input range (domain) for OF1 is 1 - 4. If the end user selects two options, OF1 is not yet bound, but its input range changes to 2 - 4. If the end user selects two more options, OF1 is bound because its Selection Count equals the number of selected options.

See SelectedCount, page 3-5.

- Instance Set: An Instance Set is bound at runtime when its Instance Count is reduced to a single value that is equal to the number of instances in the set.

The InstanceCountSystem Property is described in the *Oracle Configurator Developer User's Guide*.

- Connector: This type of node is bound at runtime when its number of connections is reduced to a single value that is equal to its Connection Count.

See ConnectionCount, page 3-6.

Additional Features of the Fusion Configurator Engine

This section lists additional FCE functionality and describes several known limitations of the Original Configurator Engine that do not exist in FCE Models.

- You can require end-users to explicitly enter a value for a Numeric or Text Feature. (The OCE allows you to require input, but only for Text Features.)

For details, see Require End-User Input Setting, page 3-19.

- The range defined for a Resource in Configurator Developer is enforced at runtime. In other words, an end user's actions cannot cause a Resource to become "over-consumed."

For details, see Totals and Resources, page 3-21.

- At runtime, Connectors can be bi-directional and may allow multiple connections.

For details, see Connectors, page 3-24.

Performance and Usability Enhancements

The FCE provides the following enhancements in the areas of performance and usability:

- **Improved modeling techniques:** The ability to specify a preferred order for executing Search Decisions and Defaults makes it easier to design and build configuration models that meet your needs.

See Specifying a Sequence for Defaults and Search Decisions, page 3-31.

- **Reduced Model maintenance:** FCE Models provide more predictable behavior than the Original Configurator Engine, making it easier to debug model design and runtime issues.

Additionally, the FCE's ability to automatically create component instances, complete a configuration, and perform complex defaulting greatly reduces the need for Configurator Extensions.

- **Improved end-user experience:** End users can see the valid input range for Numeric inputs (which results in fewer contradictions) and can allow Oracle Configurator to complete a configuration automatically. As a result, your end users can create complete and valid configurations much more quickly, with fewer errors.

See Auto-Complete, page 1-3.

Original Configurator Engine: Known Issues and Limitations

This section lists known issues and limitations of Models that use the Original Configurator Engine that do not exist in FCE Models.

- **NotTrue Logical Function:** The use of the `NotTrue` operator in Models that use the Original Configurator Engine imposes an order for rule propagation that causes configuration models to be more difficult to design and use, and may result in a "locked" state for the initial values of some items. In FCE Models, these problems do not exist because items in the configuration can never be "Unknown" and all items are bound eventually. In other words, rules that the FCE supports are never based upon the condition that an argument within a rule is unknown. Thus, the `NotTrue` operator is not necessary, and is therefore not available for use in FCE Models.

When you convert an existing Model to use the FCE, each occurrence of the `NotTrue` operator changes to `NOT`. To achieve behavior similar to the `NotTrue` operator when creating rules in an FCE Model, use the `NOT` operator.

- **Numeric Rules:** In Models that use the Original Configurator Engine, Numeric Rules can "push" (propagate) only one way: from the Operand A side of the rule to the Operand B side of the rule.

In an FCE Model, Numeric constraints propagate in both directions.

For example, a Model contains `IntegerFeatureX`, which has Minimum and Maximum Values of 0 and 20, respectively. The Model also contains the following rule:

```
(IntegerFeatureX > 10) REQUIRES BooleanFeatureY
```

In a Model that uses the Original Configurator Engine, selecting or deselecting `BooleanFeatureY` does not cause `IntegerFeatureX` to have a value greater than 10; in fact, the action has no effect on `IntegerFeatureX` at all.

In an FCE Model, selecting `BooleanFeatureY` changes the domain of `IntegerFeatureX` to '11 to 20' (this range is visible to the end user at runtime). If the end user then deselects `BooleanFeatureY`, the domain of `IntegerFeatureX` changes to '0 to 10'.

- **Comparison Rules and Intermediate Values:** When configuring a Model that uses the Original Configurator Engine, it is possible for unexpected contradictions to occur due to the use of intermediate values with Comparison Rules. This limitation is described in detail in the *Oracle Configurator Modeling Guide* (part number B13605-03). It is not possible for this scenario to occur in an FCE Model, since the FCE and the Original Configurator Engine propagate rules very differently.
- **Repetitive Rule Patterns, Redundancy, and Circular Propagation (Numeric Cycles):** For details about each issue, see the *Oracle Configurator Modeling Guide* (part number B13605-03).

Due to the manner in which rules propagate in the FCE, none of these problems can occur in an FCE Model.

- **Optional Connectors:** In Models that use the Original Configurator Engine, optional Connectors cannot participate in Logic Rules. This is not the case in an FCE Model.

See Model Node System Properties, page 3-3.

- **Modifying a Feature's Minimum and Maximum at Runtime:** In a Model that uses the Original Configurator Engine, you cannot create rules that dynamically change a Feature's Minimum and Maximum number of selections. You can create this type of rule in an FCE Model.

- **BOM Internal Quantity Computation and Visibility:** In Models that use the Original Configurator Engine, the parent node is not updated when a Numeric Rule changes the child BOM node's internal quantity, and the end user is not informed of the error.

In FCE Models, the quantity relationship between a parent BOM node and its children is always maintained when the quantity of the parent or child is modified at runtime (this is true whether the value is changed by the propagation of a rule or by the end user).

Preparing to Use the Fusion Configurator Engine

This chapter covers the following topics:

- Upgrading Oracle Configurator to use the Fusion Configurator Engine
- Configurator Preferences Page
- Profile Options
- Converting Existing Models to Use the Fusion Configurator Engine

Upgrading Oracle Configurator to use the Fusion Configurator Engine

Read this chapter if you are currently using Oracle Configurator Release 12.0 or earlier and are upgrading to release 12.1 and intend to use the FCE.

Configurator Preferences Page

For background information about this page and the various settings it contains, see "Preferences" in the *Oracle Configurator Developer User's Guide*.

When the profile option CZ: Enable Configurator Engine, page 2-3 is set to `BOTH`, then a section called Model Creation is added to the Configurator Preferences page, and this section contains a setting called Configurator Engine for New Models. Use this setting to specify a value for the profile option CZ: Configurator Engine for New Models, page 2-3, which controls which configurator engine is used when new models are subsequently created.

Profile Options

After upgrading to a version of Oracle Configurator Developer that supports the FCE (for example, R12.1), read the descriptions of the following profile options to understand how they function and decide whether the default values are appropriate

for your installation. In summary:

- **CZ: Enable Configurator Engine**, page 2-3 enables the use of the Fusion Configurator Engine and **CZ: Configurator Engine for New Models**, page 2-3 controls whether new models are associated with the FCE.
- **CZ: Use BOM Default Quantity as Domain**, page 2-6 controls the whether the Default Quantity of BOM Items is used to populate any undefined Minimum or Maximum quantities during import. If the value of that profile option is not `True`, then **CZ: Default Max Quantity Integer**, page 2-3 and **CZ: Default Max Quantity Decimal**, page 2-3 provide the values for any undefined Maximum quantities, and set the Minimum quantities (to 1 for integer items, and to 0.0 for decimal items).
- **CZ: Processing Page Delay**, page 2-3 controls how soon an informational page is displayed during long runtime operations.

Fusion Configurator Engine Profile Options

Profile Option	User	User *	Res p	Apps	Site	Required?	Default Value
CZ: Configurator Engine for New Models, page 2-3	X	X	X		X		Original
CZ: Default Max Quantity Decimal, page 2-3	X	X	X		X		1000.0
CZ: Default Max Quantity Integer, page 2-3	X	X	X		X		1000
CZ: Enable Configurator Engine, page 2-3					X		Original
CZ: Processing Page Delay, page 2-3	X	X	X		X		4000 (milliseconds)

Profile Option	User	User *	Res p	Apps	Site	Required?	Default Value
CZ: Use BOM Default Quantity as Domain, page 2-6	X	X	X		X		True

Following is a description of the symbols used in the previous table:

X: You can update the profile option at this level.

Null/no value: You cannot change the profile option value at this level.

The column marked User*, page 2-2 refers to the user-level setting made by the system administrator, as distinct from the user-level setting made by users.

Profile Options

You must set a value for profile options followed by the word "required," no default is supplied. Ordinary users can see profile options followed by the word "exposed," only system administrators can see the rest. Further details follow the list, click an item to find them.

CZ: Configurator Engine for New Models

CZ: Default Max Quantity Decimal

CZ: Default Max Quantity Integer

CZ: Enable Configurator Engine

CZ: Processing Page Delay

CZ: Use BOM Default Quantity as Domain

CZ: Configurator Engine for New Models

This profile option controls which configurator engine is used when new models are created. New models are considered to be either non-BOM Models that you create or BOM Models that you import into Configurator Developer for the first time. This profile option does not affect the configurator engine used for BOM Models that have already been imported; refreshing a BOM Model does not change the engine.

You set the value of this profile option in the Model Creation section of the Configurator Preferences Page, page 2-1. In order to be able to set this profile option, CZ: Enable Configurator Engine must be set to *Both*.

Valid values for this profile option are:

- *Original*, which causes new models to use the Original Configurator Engine

- `Fusion`, which causes new models to use the FCE

The default value is `Original`. The value of this profile option must be the same as `CZ: Enable Configurator Engine`, unless `CZ: Enable Configurator Engine` is set to `Both`, which allows you to choose either `Original` or `Fusion`.

After you create a new model, its Configurator Engine setting is displayed on the General Area of the Workbench, page 3-2.

Once a Model is created, you cannot change its associated configurator engine by changing this profile option. To change a model to use the FCE, you must use the Model Conversion Utility. For details, see *Converting Existing Models to Use the Fusion Configurator Engine*, page 2-7.

CZ: Default Max Quantity Decimal

This profile option performs the same function as `CZ: Default Quantity Integer`, page 2-3, but it provides a maximum quantity for *decimal* BOM items. The default value is 1000.0.

Note: A decimal BOM item is an item that allows an Oracle Configurator end user to enter a decimal value (such as 2.5) when specifying a quantity at runtime.

All of the information about `CZ: Default Quantity Integer` also applies to `CZ: Default Max Quantity Decimal`, including the levels at which it can be set, its interaction with `CZ: Use Default Quantity as Domain`, and so on.

CZ: Default Max Quantity Integer

This profile option specifies the integer value to assign as the Maximum Quantity for integer BOM items and Features whose Maximum Quantity is found to be blank (not defined) when you refresh or import a BOM Model, or convert an existing Model (BOM or non-BOM) to use the FCE. This profile option can be set at the User, Responsibility, and Site level. The default value is 1000.

BOM items that do not have a Maximum Quantity inherit the value of this profile option only if `CZ: Use BOM Default Quantity as Domain`, page 2-6 is `False`. If `CZ: Use BOM Default Quantity as Domain` is `True`, then this profile option has no effect.

For more information, see `CZ: Use BOM Default Quantity as Domain`, page 2-6.

Tip: Oracle strongly recommends specifying values for both the Minimum and Maximum Quantity for each BOM component in Oracle Bills of Material. For optimal performance of the Auto-Complete process, set the Maximum Quantity as low as possible and the Minimum Quantity as high as possible.

CZ: Enable Configurator Engine

This profile option controls whether the Fusion Configurator Engine is enabled for use and whether Oracle Configurator Developer displays FCE-specific content in global objects, which include UI Master Templates and UI Content Templates. This profile option can be set only at the Site level.

Valid values for this option include `Original`, `Fusion`, and `Both`. The default value is `Original`. You may want to set this option to `Both` if you need to maintain Models that were created using the Original Configurator Engine but also want to be able to create FCE Models.

Note: CZ: Enable Configurator Engine does *not* determine which configurator engine a Model uses. After upgrading to Oracle Configurator release 12.1, all existing Models continue to use the Original Configurator Engine by default. For details about the profile option that controls whether new Models use the FCE, see CZ: Configurator Engine for New Models, page 2-3. All *existing* BOM and non-BOM Models must be converted if you want them to use the FCE. For details, see Converting Existing Models to Use the Fusion Configurator Engine, page 2-7.

If this profile option is set to either `Fusion` or `Both`, then the FCE is available for use and the following content is visible in Configurator Developer:

- Configurator Engine setting (appears in the Create Model page and the General area of the Workbench when you open a Model for editing)
For details about this setting, see General Area of the Workbench, page 3-2.
- User Interface Content Templates that are available only with the Fusion Configurator Engine
- FCE-specific System Properties (Model node and configuration session System Properties) in Models and Content Templates
- FCE-specific settings in UI Master Templates and the User Interface Definition page

When you set this profile option to `Both`, the Configurator Preferences page includes a setting that enables you to specify a value for the profile option CZ: Configurator Engine for New Models. For more information, see:

- Configurator Preferences Page, page 2-1
- CZ: Configurator Engine for New Models, page 2-3

CZ: Processing Page Delay

Background: An Oracle Configurator Developer user can choose to display a processing page for UI actions that take a long time to complete. An example of a UI action that can cause a processing page to be displayed is Auto-Complete Configuration. For details, see *Displaying a Processing Page at Runtime*, page 3-64.

The profile option CZ: Processing Page Delay specifies how many seconds Oracle Configurator waits after an end user performs an action before displaying a "processing request" page. If the end user's request completes before the specified time expires, Oracle Configurator does not display this page.

The default value of this profile option, specified in milliseconds, is 4000 (4 seconds), and it can be set at the User, Responsibility, and Site level.

Important: Oracle strongly recommends that you specify a value of no more than 8 (seconds) for this profile option. Additionally, be sure that the value of this profile option is less than your Web server and browser time out settings. If the Web server or browser time out setting is less than the profile option, then the profile option will have no effect at runtime.

CZ: Use BOM Default Quantity as Domain

In Oracle Bills of Material, a user must enter a Default Quantity when defining a BOM item, but specifying a Minimum and Maximum Quantity is optional. When you import, refresh, or convert a Model to use the FCE, any blank occurrences of those quantities are automatically set to a value, either by this profile option or by CZ: Default Max Quantity Integer, page 2-3 and CZ: Default Max Quantity Decimal, page 2-3.

This profile option allows you to apply a common interpretation of blank Minimum and Maximum Quantities that limits a BOM item's domain and improves performance of the Auto-Complete process at runtime.

Note: The term "domain" is defined in *Domain Ordering Setting*, page 3-16.

When this profile option is set to `True`, any undefined Minimum or Maximum Quantities are set to the item's Default Quantity when you import, refresh, or convert a Model to use the FCE. For example, an item's Minimum Quantity is 1, its Maximum Quantity is blank, and its Default Quantity is 1. When you import, refresh or convert the item's parent BOM Model to use the FCE, the item's Maximum Quantity is set to 1.

When this profile option is set to `False` and you import, refresh, or convert a BOM Model to use the FCE:

- The profile options CZ: Default Max Quantity Integer, page 2-3 and CZ: Default

Max Quantity Decimal, page 2-3 provide values for any BOM items that do not have a Maximum Quantity (which profile option is used depends on whether the BOM item accepts a decimal or integer quantity at runtime).

- Any Minimum Quantities that do not have a value are set to 1 (for integer items) or 0.0 (for decimal items).

This profile option can be set at the User, Responsibility, and Site level.

Tip: Oracle recommends that all BOM components have values specified for the Minimum and Maximum Quantity settings.

Converting Existing Models to Use the Fusion Configurator Engine

This section contains the following topics:

- Introduction, page 2-7
- Model Conversion Utility Output, page 2-7
- Model Conversion Utility Report, page 2-12
- Running the Model Conversion Utility, page 2-14

Introduction

If you are upgrading to Release 12.1 or later from a previous version of Oracle Configurator, you must run the Model Conversion Utility to convert any existing Models that you want to use the Fusion Configurator Engine. This utility is an Oracle Applications concurrent program that is available when you log in to Oracle Applications using either the Oracle Configurator Developer or Oracle Configurator Administrator responsibility.

The Model Conversion Utility creates a copy of a Model that was created using the Original Configurator Engine (the source Model) and then modifies the copied Model's structure, rules, and related UI objects so they are compatible with the FCE. The utility does not make any changes to the source Model.

If you are installing Oracle Configurator for the first time and you want to use the FCE and create FCE Models by default, you need only verify that the profile options CZ: Enable Configurator Engine and CZ: Configurator Engine for New Models have the appropriate values. For details, see Profile Options, page 2-1.

Model Conversion Utility Output

The Model Conversion Utility generates a report that describes in detail how each

Model node, rule, and related UI object changed during the conversion process. Some messages prompt you to review specific areas of the Model or make changes to ensure the converted Model performs as expected at runtime. For details, see Model Conversion Utility Report, page 2-12. The sections below describe at a high level how the Model Conversion Utility converts a Model's Structure, page 2-8, Rules, page 2-10, and associated User Interface objects, page 2-12.

After the Model Conversion Utility completes successfully, you can access the converted Model in the same Folder as the source Model. A converted Model has the same name as the source Model, but the name is appended with "-[FCE]". For example, "Custom Sentinel Desktop - [FCE]". If a Model with the same name already exists (for example, if a Model is converted more than once), then an ordinal number is appended to the name. For example, "Custom Sentinel Desktop - [FCE][2]".

Model Structure

Following is a summary of how each type of Model node changes when you convert a Model to use the FCE. Details about each change are provided in the Model Conversion Utility Report, page 2-12.

- **Boolean Features:**
 - Initial Values are converted to Requires Logic Rules (Rule Class is set to Default)
- **Totals and Resources:**
 - Converted to Decimal Totals and Decimal Resources

For optimal performance of the Auto-Complete process, consider re-creating all converted Totals and Resources as Integer Totals and Integer Resources. For details, see Totals and Resources, page 3-21.
 - Effectivity settings are removed (this includes date ranges, Effectivity Sets, and Usages)
 - Initial Values are expressed as constant terms in new Accumulator Rules.

Note: When you generate Model logic, this term is rolled into the sum constraint that accumulates all other rules that contribute to or consume from the node. Additionally, if the Total or Resource is in a Model that is referenced by another Model, and the node participates in any rules that belong to the referencing Model, the utility creates an additional Accumulator Rule containing the Initial Value in each ancestor Model.

- **Numeric Features:**

- Default values are provided if Minimum or Maximum is null.

The Model Conversion Report prompts you to review the new values.

- Initial Values are expressed as constant terms in new Accumulator Rules

Note: The "Additional Information" paragraph above (in the section describing Totals and Resources) also applies to Numeric Features.

- Effectivity settings are removed (this includes date ranges, Effectivity Sets, and Usages)

- **Counted Option Features:** Maximum Quantity per Option setting (with default value) is added

- **BOM Items:** Default values are provided if Minimum or Maximum Quantity is blank

The utility provides a value of 1 for the Minimum Quantity (0.0 for decimal BOM items) and a large number for the Maximum Quantity. For more information, see BOM Nodes, page 3-22.

- **Model References:** For Multi-Instantiable BOM Model References, the settings for Initial Minimum and Maximum Instances are removed. The BOM Maximum Quantity setting now defines the total Quantity allowed across all Instances.

- **Components:** Effectivity settings are removed for required, single-instance (1/1) Components (this includes date ranges, Effectivity Sets, and Usages).

Optional single-instance Components and Components that allow multiple instances are not modified during the conversion.

- **Connectors:**

Important: Connectors are not supported in the current release of the Fusion Configurator Engine. The information about Connectors provided in this document is only for planning purposes. However, Connectors continue to be supported for models using the Original Configurator Engine.

- Minimum Target Instances and Maximum Target Instances settings are added (default values are 0/1)

- Connection Required setting is removed

For additional information about the changes and terms used above, refer to the following sections:

- Initial Values, page 3-21
- Effectivity, page 3-26
- Connectors, page 3-24

Rules

By default, all rules created by the Model Conversion Utility appear in the Rules area of the Workbench in a Folder called "Rules Generated by Model Conversion." This Folder appears as a child of the root Rules Folder. You can leave the rules in this Folder, or move them to a different location.

The Model Conversion Utility creates new rules to replace rules and settings that do not map directly to the FCE. For example, the utility creates Accumulator Rules to replace Numeric Rules and Initial Value settings. See Initial Values, page 3-21.

Following is a summary of the types of rules that the FCE does not support, and how some rules and Model node settings change during the conversion process:

- The FCE does not support Numeric Rules in the same way as the Original Configurator Engine. Therefore, all Numeric Rules (and Statement Rules that use either the 'CONTRIBUTE' or 'CONSUMES' operators) are converted to Accumulator Rules, page 3-32.

In Statement Rules, the CONTRIBUTE...TO operator changes to ADD...TO during the conversion process, and CONSUMES...FROM changes to SUBTRACT...FROM. The resulting rule syntax is:

```
ADD OptionA TO OptionB
SUBTRACT OptionX FROM OptionY
```

In both cases, the utility removes the rule's violation message. Configurator Developer displays a standard, predefined message when the Accumulator Rule is violated at runtime.

- All Logic Rules that use the Defaults operator (Defaults Logic Rules) are converted to Statement Rules that use the IMPLIES operator, and these rules are assigned a Rule Class of "Default'.

For example, before converting a Model, a Defaults Logic Rule has the following definition:

```
OptionA DEFAULTS AnyTrue (OptionB, Option C)
```

After the conversion, the Model contains the following "Default" rule:

```
AnyTrue ('OptionA') IMPLIES AnyTrue ('OptionB', 'OptionC')
```

- All expressions in Logic Rules in which Count Features participate are converted so that the Count Features become greater-than-0 (zero) expressions. For example, the source Model has the following rule:

```
CountFeatureX IMPLIES BooleanFeatureY
```

After the conversion, the Model contains the following rule:

```
(CountFeatureX > 0) IMPLIES BooleanFeatureY
```

- When converting Compatibility Rules (Explicit Compatibility Rules, Property-based Compatibility Rules, and Design Charts), a Failure message appears in the Model Conversion Report if more than one participant in the rule has a maximum number of selections greater than 1.

The rule conversion fails. The Failure message suggests modifying the rule.

- When converting Compatibility Rules, a Failure message is generated if a BOM Model is a participant in the rule.
- The FCE does not support the `ATAN2` operator. During the conversion, each occurrence of this operator changes to `ATAN`.
- The FCE does not support the `NOTTRUE` operator. During the conversion, each occurrence of this operator changes to `NOT`.
- For each Configurator Extension that is bound to the `onValidateEligibleTarget` event, the utility removes the event binding and generates a Warning message in the Model Conversion Report.
- For any Configurator Extension that is bound to an event that could be triggered during Auto-Complete, the utility generates an Advisory message stating that the event will *not* be triggered, and the CX will not execute during Auto-Complete. These events are:
 - `postValueChange`
 - `postInstanceAdd`
 - `postInstanceDelete`
 - `postConnect`
 - `postDisconnect`

- For any Configurator Extension that is bound to an event that could be triggered after Auto-Complete, the utility generates an Advisory message stating that the CX may fail if it attempts to modify a completed configuration (if not in Adjust Mode).
- The FCE does not support Functional Companions. The utility generates a Warning.

User Interface Objects

After the Model Conversion Utility completes successfully, the UI Definition for each of the converted Model's UIs refer to FCE-specific icons, Message Templates and Utility Templates. All custom and predefined UI Master Templates also refer to these icons and templates, and they contain additional settings for displaying BOM and non-BOM content.

For more information about these changes, see:

- User Interface Definition, page 3-67
- User Interface Master Templates, page 3-48

The Model Conversion Utility does not modify any other UI objects, but it does review all pages, elements and custom UI Content Templates for potential incompatibilities with the FCE. For example, the utility changes the `Unsatisfied System Property` to `InputRequired`. As a result, any runtime conditions in which `InputRequired` is used may appear or behave differently after conversion. All other incompatibilities are described in detail in the Model Conversion Utility Report, page 2-12.

Model Conversion Utility Report

The Model Conversion Utility generates this report each time you convert one or more Models to use the Fusion Configurator Engine. The report is in XML format and Oracle provides an XML Publisher template for formatting the output. You can optionally define additional XML Publisher templates if you want to customize the report format.

To learn how to run the Model Conversion Utility and view the Model Conversion Report, see *Running the Model Conversion Utility*, page 2-14.

For each Model that was converted, the report summarizes the conversion process by listing:

- The names of the source and converted Model(s)
- The path to the location of both the source and the converted Model(s)
- How many messages of each type were generated during the conversion process

There are four types of messages that may appear in the Model Conversion Utility report. Following is a description of each type:

- **1 - Failure:** The specified object could not be converted. To resolve this type of error, you must either modify or delete the object in Configurator Developer.

For example, when you convert a Model that has an invalid rule, the report contains a Failure message similar to the following:

```
Rule contains invalid syntax; not converted. Please review and
correct, or delete.
```

- **2 - Warning:** Converting the specified object may have introduced unexpected or undesirable behavior in the Model. Reviewing the object in Configurator Developer is strongly recommended.

For example, after converting a Defaults Logic Rule, the report contains a Warning message similar to the following:

```
DEFAULTS operator not supported; Rule has been converted to a
Default Rule using Implies operator.
```

- **3 - Advisory:** The object converted successfully, but you may be able to improve performance by making additional changes in Configurator Developer. Reviewing the object is recommended.

For example, after converting a Numeric Feature, the report contains an Advisory message similar to the following:

```
Maximum Value set to 1000. Consider specifying a lower bound if
possible.
```

In this example, the value is the default provided by the profile option CZ: Default Max Quantity Integer, page 2-3.

- **4 - Information:** The object converted successfully with no issues, and reviewing the object in Configurator Developer is optional.

For example, after converting a Boolean Feature that has an Initial Value, the report contains an Information message similar to the following:

```
Initial Value removed; replaced by Default Rule RuleName If value
is not required prior to completion of the configuration, consider
changing Rule to a Search Decision for better performance.
```

Model Conversion Report: Example

Summary

Models Converted						
Source Model	Converted Model	Folder Path	Messages			
			Failure	Warning	Advisory	Information
M1	M1-[FCE]	Folder X/Folder Y	0	3	7	36
M2	M2-[FCE]	Folder X/Folder Y/Folder Z	1	5	29	112
M3	M3-[FCE]	Folder X/Folder Y/Folder Z	0	8	31	54

UI Templates Scanned

Template Name	Folder	Warning Messages
Custom Checkbox Template	UI Templates/Content Templates/Custom Templates	2
Custom Dropdown Template	UI Templates/Content Templates/Custom Templates	1

Model Conversion Messages

M1-[FCE] in Folder X/ Folder Y

Structure

Message Type	Node Name	Path	Message
3-Advisory	Numeric Feature 1	Root.'Component 1'.Numeric Feature 1'	Minimum Value set to 0. Consider specifying a higher bound if possible.
3-Advisory	Numeric Feature 1	Root.'Component 1'.Numeric Feature 1'	Maximum Value set to 65535. Consider specifying a lower bound if possible.

Rules

Message Type	Rule Name	Folder	Message
2-Warning	Statement Rule 22	Folder 2/Statement Rule 22	ATAN2 operator not supported; converted to ATAN. Please review and ensure that division by zero cannot occur.
2-Warning	Logic Rule 7	Folder 1/Folder 2/Logic Rule 7	DEFAULTS operator not supported; Rule has been converted to a Default Rule using Implies operator.

User Interface

Message Type	Element Name	Path	Message
2-Warning	Image X	UI 1/Pages/Page 1/Row Layout 2/Image X	System Property 'Unsatisfied' in Display Condition converted to 'UserInputRequired'. Runtime appearance and/or behavior may be affected.

Running the Model Conversion Utility

Procedure

To convert Models to use the FCE:

1. Log into Configurator Developer using either the Oracle Configurator Developer or Oracle Configurator Administrator responsibility.
2. In the Main area of the Repository, select the Model(s) that you want to convert.
Ensure that none of the selected Models is locked. See Locking Considerations, page 2-15
3. From the Actions list, select **Convert Model(s) to use FCE**, and then click **Go**.
4. Review the list of selected and related Models, and then click either **Convert All** or **Convert Selected**.

Important: If any related Models are locked, Oracle recommends canceling the conversion and then resubmitting it later (when all related Models are unlocked) to preserve any shared Model references.

5. Configurator Developer displays a message similar to the following:

```
Created Conversion Set with ID RequestID. Please review the output
of the Model Conversion concurrent process for important messages
about the conversion.
```

Make note of the Request ID (it is required in a subsequent step), and then click **OK**.

6. Exit Configurator Developer and return to the E-Business Suite Home page.
7. Select **Concurrent Programs > Schedule**, and then enter either "Process a Single Model Conversion" or "Process Pending Model Conversions" in the **Program Name** field.

Tip: Click the list of values icon to search for the program name.

8. If you entered "Process a Single Model Conversion" in the previous step, enter the Model Conversion Set ID.
9. Click **Next** and then enter any optional request parameters. For example, enter Scheduling, Layout, Notifications, and Printing options.

After submitting the request, the Requests page appears. When the Model Conversion concurrent program has completed successfully, click the icon in the Output column to review the Model Conversion Report. Review this report carefully, as it describes in detail how the Model and its rules changed during the conversion and may prompt you to make additional changes in Configurator Developer. For details, see Model Conversion Report, page 2-12.

Locking Considerations

When submitting a job to the Model Conversion Utility, none of the Models that you select for conversion can be locked by another user. If one or more of the selected Models are locked, Configurator Developer displays a message similar to the following:

```
Some of the Models selected for conversion are locked by another user.
Conversion cannot be completed.
```

In this case the only option is to cancel the conversion, but you can resubmit the process later when the Models are unlocked.

The Model Conversion Utility also checks whether the following Models are locked:

- Models that the selected Model(s) reference. (These are also shown as selected models, since they are implicitly selected.)
- Models that reference the selected Model(s). (Also included are models that they reference, models that reference them, and so on.)

If none of the related Models are locked by another user, Oracle recommends that you

click **Convert All** to convert the source Models and all related Models at the same time. This ensures all shared Model references are preserved during the conversion.

If any of the related Models are locked, then Configurator Developer lists them and displays a message similar to the following:

Some of the Related Models are locked by another user. Conversion of the Selected Models is possible with loss of some shared references.

In this case, you can either cancel the operation or convert only the Models that you selected for conversion. If you choose to convert only the selected Models, the related Models are not converted.

Building a Configuration Model Using the Fusion Configurator Engine

This chapter covers the following topics:

- Model Structure
- Configuration Rules
- Creating and Editing a User Interface
- Unit Testing a Configuration Model Using the Model Debugger

Model Structure

This section describes Model structure settings and characteristics that are available only when using the FCE. It includes the following sections:

- General Area of the Workbench, page 3-2
- Model References, page 3-3
- Properties, page 3-3
- Domain Ordering Setting, page 3-16
- Require End-User Input Setting, page 3-19
- Text Features, page 3-19
- Totals and Resources, page 3-21
- Initial Values, page 3-21
- BOM Nodes, page 3-22

- Connectors, page 3-24
- Effectivity, page 3-26

General Area of the Workbench

For background information about the General area of the Workbench, see the *Oracle Configurator Developer User's Guide*.

Model Details: Configurator Engine

When the profile option CZ: Enable Configurator Engine is set to either `Fusion` or `Both`, the General area of the Workbench includes the Configurator Engine setting. This setting is specific to the Model you are viewing or editing, is read-only, and it can have a value of either `Original` or `Fusion`.

If the value of this setting is `Fusion`, then all FCE-related content and functionality is available when:

- Viewing or editing the Model's structure, rules, and User Interface(s) in Configurator Developer
- Configuring the Model in a runtime UI, or the Model Debugger

A value is determined for this setting when you create a Model (either manually in Configurator Developer, or by importing a BOM Model), or convert an existing Model to use the FCE by running the Model Conversion Utility.

The value of this setting is `Original` if the Model was created or imported:

- In a version of Configurator Developer that does not support the FCE
- When the FCE was not enabled, or
- When the profile option CZ: Configurator Engine for New Models was set to `Original` (see CZ: Configurator Engine for New Models, page 2-3).

The value of this setting is `Fusion` if the Model was:

- Created or imported when the profile option CZ: Configurator Engine for New Models was set to `Fusion`
- or
- Converted to use the FCE

In this case, the Configurator Engine setting is `Original` in the source model and `Fusion` in the converted model.

For more information, see *Converting Existing Models to Use the Fusion Configurator Engine*, page 2-7.

Note: Once it is determined that a Model uses the FCE, changing the value of CZ: Configurator Engine for New Models or CZ: Enable Configurator Engine does not affect the Model's runtime behavior, or the availability of FCE-specific content when editing the Model in Configurator Developer.

In the Main area of the Repository, you can create a personalized view that includes the column Configurator Engine Type. This column displays the engine type associated with each Model: *Original* or *Fusion*.

Model Report

The Model Report includes features that are particular to the FCE. For background information about the Model Report, see the *Oracle Configurator Developer User's Guide*.

Model References

When working in an FCE Model, you cannot create a Model Reference to a Model that does *not* use the FCE. (The reverse is also true.) For this reason, Models that do not use the same configurator engine as the Model you are editing are not available for selection.

Tip: For optimal performance of the Auto-Complete process, Oracle strongly recommends setting the Maximum Instances as low as possible and the Minimum Instances as high as possible for all BOM and non-BOM Model Reference nodes.

For more information about BOM Model References, see BOM Nodes, page 3-22.

Properties

This section describes properties that are available only when building a configuration model that uses the FCE in Configurator Developer, and when configuring such a model at runtime. It includes the following sections:

- Model Node System Properties, page 3-3
- Configuration Session Properties, page 3-11

Model Node System Properties

This section describes the System Properties that are available only to FCE Models and indicates the type of node(s) with which each Property is associated.

Almost all System Properties can be used when defining text expressions and runtime conditions for UI elements. Exceptions include System Properties that return objects or collections, such as `Children` and `SummaryChildren`. For details about these System Properties, or general information about text expressions and runtime conditions, see the *Oracle Configurator Developer User's Guide*.

Some System Properties are also **mutable**, which means that the Property's value can be directly set by the propagation of a rule or by the end user at runtime. Examples of nodes that are mutable include `InstanceCount`, `ConnectionCount`, `RelativeQuantity`, and `SelectedCount`. In the System Properties, page 3-5 table, the Description column indicates whether or not each Property is mutable.

Where a set of System Properties defines a Minimum and Maximum value, count, or quantity (for example, `MinimumSelected` and `MaximumSelected`), only the node's *current* value is mutable. In these cases, the minimum and maximum values keep track of the node's current *domain* at runtime, not its current value.

Example

For example, if you want to be sure the value of Feature X does not fall below 5 at runtime, define a rule like this:

```
FeatureX.Value() >= 5
```

Example

Because the `MinValue` System Property is not mutable, you cannot use it to constrain a node's minimum value. Therefore, the following rule is invalid:

```
FeatureX.MinValue() = 5
```

You can define rules in which optional Connectors participate by using the `ConnectionCount` System Property.

Example

For example, to prevent an optional Connector from being connected at runtime, define a rule like this:

```
NOT(OF1) REQUIRES (ConnectorA.ConnectionCount = 0)
```

In this example, an end user cannot connect Connector A when OF1 is deselected or excluded.

To require an end user to connect an optional Connector at runtime, define a rule like this:

```
OF1 REQUIRES (ConnectorA.ConnectionCount > 0)
```

In this example, Oracle Configurator requires Connector A to be connected when OF1 is selected.

Example

To dynamically modify how many instances of a component can be created at runtime, use the `InstanceCount` System Property.

For example:

```
OF1 IMPLIES ComponentX.InstanceCount() > 2
```

```
OF1 IMPLIES ComponentX.InstanceCount() < 10
```


The first rule specifies that more than 2 instances of ComponentX will be created when OF1 is selected. In this case, the rule modifies the minimum number of instances specified in Configurator Developer.

The second rule specifies that no more than 9 instances of ComponentX will be created when OF1 is selected. In this case, the rule modifies the maximum number of instances specified in Configurator Developer. Note that ComponentX must have at least 2 instances to satisfy the constraints defined in this example.

You can also control the relative or absolute quantity of a BOM node using Constraints.

Example

For example:

```
OF1 IMPLIES BOMOptionClassX.Quantity() < 7
```

```
OF2 IMPLIES BOMOptionClassY.RelativeQuantity() > 10
```

System Properties

System Property	Description	Node Type(s)	Returns
MinRelQuantity	The node's minimum relative quantity. Not mutable.	BOM nodes	Integer or Decimal
MaxRelQuantity	The node's maximum relative quantity. Not mutable.	BOM nodes	Integer or Decimal
RelativeQuantity	Quantity of a BOM node relative to the quantity of its parent. Mutable.	All BOM nodes	Integer or Decimal
SelectedCount	The number of unique selected children for an Option Feature. Mutable.	Option Feature	Integer
MinConnections	Minimum number of target instances the Connector should have. Not mutable.	Connectors	Integer

System Property	Description	Node Type(s)	Returns
MaxConnections	Maximum number of target instances the Connector may have. Not mutable.	Connectors	Integer
ConnectionCount	Number of target instances assigned to a Connector. Mutable.	Connectors	Integer
DefinitionMinValue	The node's minimum value as defined in the Model. Not mutable.	All nodes that have a Minimum and Maximum value.	Integer or Decimal
DefinitionMaxValue	The node's maximum value as defined in the Model. Not mutable.	All nodes that have a Minimum and Maximum value.	Integer or Decimal
DefinitionMinQuantity	The node's minimum quantity as defined in the Model. Not mutable.	All nodes that have a Minimum and Maximum Quantity (for example, all BOM nodes).	Integer or Decimal
DefinitionMaxQuantity	The node's maximum quantity as defined in the Model. Not mutable.	All nodes that have a Minimum and Maximum Quantity (for example, all BOM nodes).	Integer or Decimal
DefinitionMinRelQuantity	The node's minimum relative quantity, as defined in the Model. Not mutable.	All BOM nodes	Integer or Decimal

System Property	Description	Node Type(s)	Returns
DefinitionMaxRelQuantity	<p>The node's maximum relative quantity, as defined in the Model.</p> <p>Not mutable.</p>	All BOM nodes	Integer or Decimal
DefinitionMinSelected	<p>Minimum number of selections as defined in the Model.</p> <p>Not mutable.</p>	All nodes that have a Minimum and Maximum Selections (for example, BOM Models, BOM Option Classes, and Option Features)	Integer
DefinitionMaxSelected	<p>Maximum number of selections as defined in the Model.</p> <p>Not mutable.</p>	All nodes that have a Minimum and Maximum Selections (for example, BOM Models, BOM Option Classes, and Option Features)	Integer
DefinitionMinInstances	<p>Minimum number of instances as defined in the Model.</p> <p>Not mutable.</p>	All nodes that have a Minimum and Maximum Instances (for example, Components and Model References)	Integer

System Property	Description	Node Type(s)	Returns
DefinitionMaxInstances	<p>Maximum number of instances as defined in the Model.</p> <p>Not mutable.</p>	All nodes that have a Minimum and Maximum Instances (for example, Components and Model References)	Integer
DefinitionMinConnections	<p>Minimum number of target instances for a Connector, as defined in the Model.</p> <p>Not mutable.</p>	Connectors	Integer
DefinitionMaxConnections	<p>Maximum number of target instances for a Connector, as defined in the Model.</p> <p>Not mutable.</p>	Connectors	Integer
Proposed	<p>Returns True if the current state or value of the node is the result of a Default or Search Decision.</p> <p>Not mutable.</p>	All nodes that support state, value, or quantity.	Boolean

System Property	Description	Node Type(s)	Returns
IsBound	<p>Returns True if the node is completely bound. This means both domain and cardinality, if applicable.</p> <p>Note: For nodes with both state and quantity, both must be bound. For BOM nodes, both absolute and relative quantity must be bound.</p> <p>Not mutable.</p>	<p>All variables: Numeric Features; Option Features; Boolean Features; BOM Models and Option Classes; Totals and Resources; instance sets; Connectors</p>	Boolean
IsBoundSelectionState	<p>Returns True if the state of the node is bound.</p> <p>Not mutable.</p>	<p>All state variables, which include, Option Features, Options, Boolean Features, BOM Models, BOM Option Classes, and BOM Standard Items</p>	Boolean
IsBoundQuantity	<p>Returns True if the (absolute) quantity of the node is bound.</p> <p>Not mutable.</p>	<p>All variables with (absolute) Quantity, which includes, BOM Models, BOM Option Classes, and BOM Standard Items</p>	Boolean

System Property	Description	Node Type(s)	Returns
IsBoundRelQuantity	<p>Returns True if the relative quantity of the node is bound.</p> <p>Not mutable.</p>	All variables with relative Quantity, which include BOM Models, BOM Option Classes, and BOM Standard Items.	Boolean
InputRequiredFlag	<p>Returns True if a node is marked 'User Input Required' in Configurator Developer (or by other means such as a Configurator Extension), regardless of whether or not it is bound. (Related to display of Input Required indicator.)</p> <p>Not mutable.</p> <p>See Require End User Input Setting, page 3-19.</p>	All nodes, although some cannot be set to True.	Boolean
InputRequired	<p>Returns True if a node is marked 'User Input Required' in Configurator Developer (or by other means such as a Configurator Extension) and is <i>not</i> bound. (Related to display of Input Required indicator.)</p> <p>Not mutable.</p> <p>See Require End User Input Setting, page 3-19.</p>	All nodes, although some cannot be set to True.	Boolean

System Property	Description	Node Type(s)	Returns
InputRequiredInSubtree	Returns True if <code>InputRequired</code> is True on this node or a descendant. Not mutable.	All nodes.	Boolean
InputRequiredError	Returns True when <code>node.InputRequired</code> is True and <code>Session.InErrorMode()</code> is True. Not mutable.	All nodes.	Boolean
ChangedByAC	Returns True if the selection state or quantity of the node was changed as a result of the most recent Auto-Complete. Not mutable.	All nodes	Boolean
ValidationErrorText	Returns text set by a Configurator Extension in the event of a validation error on the node. Not mutable.	All nodes	String

Configuration Session Properties

This section lists properties that refer to the state of a runtime configuration session and are not associated with Model nodes. There are two categories of configuration session properties that are available with the FCE:

- Configuration Status
- Conflict Processing

Refer to the tables below for details.

Configuration Session Properties: Configuration Status

Name	Description	Data Type	Context
ConfigComplete	Returns True if the configuration is complete: <ul style="list-style-type: none">• No items exist with <code>InputRequired = True</code>• Configuration is valid• Configuration is bound	Boolean	Global
HasItemsToAddress	Returns True if any of the following conditions exist: <ul style="list-style-type: none">• <code>Session.InputRequired</code> is True• <code>Session.Valid</code> is False	Boolean	Global
InputRequired	Returns True if the configuration contains any items with <code>InputRequired = True</code> .	Boolean	Global
InErrorMode	Initially False; updated on each request for Finish or Auto-Complete. Set to True if <code>Session.InputRequired()</code> is True. Set to False if <code>Session.InputRequired()</code> is False.	Boolean	Global

Name	Description	Data Type	Context
AutoCompleteSuccessful	<p>Returns False before first Auto-Complete in a session; returns True when Auto-Complete succeeds in binding the configuration and False when it fails. Reverts to False on Undo Auto-Complete or Adjust Configuration. The state persists until the next Auto-Complete.</p> <p>Note: Unbound Text Features that are User Input Required are ignored in computing this property. This means it is possible for AutoCompleteSuccessful to be True but ConfigComplete to be False.</p>	Boolean	Global

Name	Description	Data Type	Context
ConfigurationChangedByAC	Returns True if Auto-Complete resulted in changes to the orderable items in the configuration (or their attributes) and/or the addition of an unbound User Input Required Text Feature and/or the configuration is invalidated by a post-Auto-Complete Configurator Extension. (Note: Changes in numeric attribute values from unbound to zero are ignored.) Reverts to False on Undo Auto-Complete. The state persists until the next Auto-Complete.	Boolean	Global
InAdjustMode	Returns True if the configuration is in Adjust Mode (has Auto-Complete decisions in the request queue). Reverts to False on Undo Auto-Complete action.	Boolean	Global
ListPriceChangedByAC	Returns True if TotalListPrice was changed as a result of the most recent Auto-Complete.	Boolean	Global

Name	Description	Data Type	Context
SellingPriceChangedByAC	Returns True if TotalSellingPrice was changed as a result of the most recent Auto-Complete.	Boolean	Global
ATPChangedByAC	Returns True if ATPRollup was changed as a result of the most recent Auto-Complete.	Boolean	Global

Configuration Session Properties: Conflict Processing

Name	Description	Data Type	Context
AutoOverrideEnabled	<p>Returns True if Auto-Override is enabled, False if it is disabled. The value is derived from the Auto-Complete for Conflicts setting in the UI Definition.</p> <p>For more information, see Auto-Override, page 4-10</p>	Boolean	Global

Name	Description	Data Type	Context
ConflictOverridable	Indicates whether the current conflict is overridable. Returns True when processing a primary conflict that was triggered by a User Request, or when an empty explanations list is returned for a secondary conflict where the primary conflict has a User Request.	Boolean	Only required during conflict processing; False at other times.
OverrideSuccessful	Returns True in a Conflict flow if Override was attempted and it succeeded. State persists until next Conflict occurs.	Boolean	Global, but generally only used in templates related to Conflict processing.

Selection State

For details about how Oracle Configurator uses System Properties such as `SelectionMode` to display the selection state of options at runtime, see the *Oracle Configurator Developer User's Guide*. This section lists System Property values that indicate an option's selection state that are available only in FCE Models.

At runtime, the value of the `SelectionMode` and `DetailedSelectionMode` System Properties is `Selectable` for options that are neither selected nor excluded from a configuration.

In FCE Models, the values of the `DetailedSelectionMode` System Property also include `Recommended (Proposed Selected)` and `Not Recommended (Proposed Excluded)`. These values refer to options that were added to or excluded from the configuration by Auto-Complete.

For more information, see *Logic State Display*, page 4-1.

Domain Ordering Setting

In Oracle Configurator, the term "domain" refers to the range of possible values that either the end user or the Auto-Complete process can provide for an option at runtime.

For example, you specify a domain for Numeric Features in Configurator Developer by specifying Minimum and Maximum values. For example, the domain for an Integer Feature could be "1 - 100," which means a value of 1 through 100, inclusive.

Note: At runtime, a variable's domain may be dynamically reduced by the propagation of constraints. This is explained in Domain Display and Availability, page 4-1.)

Use the Domain Ordering setting to control how Auto-Complete determines a value for Numeric Features that do not yet have a value, and Boolean Features that are still Unbound. (An Unbound Boolean Feature is one that is neither True nor False by default, and there has been no decision made about it yet during the configuration session.)

Auto-Complete refers to the Domain Ordering setting, as well as other constraints and preferences that you have defined in the Model, to reduce an option's domain and determine the value (or selection state) that you want it to have in a completed configuration.

Important: For optimal performance of the Auto-Complete process, set all domain ranges to be as narrow as possible. In other words, set minimum values as high as possible and maximum values as low as possible.

The available Domain Ordering settings vary by node type.

Integer Features:

- **System Default:** This is the default setting for Integer, Decimal, and Boolean Features. Accept this setting if you want Auto-Complete to use its own default method for determining a value. This method provides optimal runtime performance and is recommended if you do not have a preference for the node's value or selection state at runtime.
- **Linear Search, Min to Max:** Select this setting if you want Auto-Complete to apply values within the specified domain in increasing order, beginning with the node's Minimum value.
- **Linear Search, Max to Min:** Select this setting if you want Auto-Complete to apply values within the specified domain in decreasing order, beginning with the node's Maximum value.
- **Binary Search, Decreasing Max:** Select this setting if you want Auto-Complete to successively split the domain in a binary search until either a single value is bound, or no solution is found. When you select this setting, Auto-Complete checks the lower half of the domain first.

- **Binary Search, Increasing Min:** This setting is similar to "Binary Search, Decreasing Max", except that when you select this setting, Auto-Complete checks the upper half of the domain first.

Decimal Features:

- **System Default:** See "System Default" for Integer Features, above.
- **Binary Search, Decreasing Max:** See "Binary Search, Decreasing Max" for Integer Features, above.
- **Binary Search, Increasing Min:** See "Binary Search, Increasing Min" for Integer Features, above.

UI Templates for Numeric Feature Domain Display:

By default, the following UI Content Templates display a Numeric Feature's domain at runtime:

- Decimal Input with Range Display, page 3-50
- Integer Input with Range Display, page 3-50

Boolean Features:

- **System Default:** See "System Default" for Integer Features, above.
- **Prefer False:** Select this setting if you want Auto-Complete to set the Boolean Feature to False (deselected) first and then, if it cannot be set to False, set it to True (select it).
- **Prefer True:** Select this setting if you want Auto-Complete to set the Boolean Feature to True (selected) first, and then, if it cannot be set to True, set it to False (deselect it).

Domain Ordering and Other Node Types

The domain for Model References and Connectors is the set of existing or possible instances of the referenced or target Model, along with the minimum and maximum instances.

The Domain Ordering setting is not available for Option Features and BOM Option Classes. The Auto-Complete process selects options from these nodes based on the order in which they appear in the Model structure. For example, Auto-Complete selects the first Option that appears below its parent Option Feature in the Model structure.

The domain for Option Features and BOM Option Classes is the set of child nodes, along with the minimum and maximum number of selections specified for the parent nodes in Configurator Developer.

Require End-User Input Setting

Use the Require End-User Input setting to indicate that a Feature, Connector, or BOM Option Class must be bound by the end user, rather than by Auto-Complete, at runtime. Select this setting only if it is important for the end-user to explicitly enter a value, create a connection, or make a selection at runtime.

Note: Auto-Complete cannot run until the end user binds all variables that require end-user input.

The Require End-User Input setting is available for the following node types:

- Numeric Features
- Text Features
- Option Features
- Connectors
- BOM Option Classes

For BOM Option Classes and Option Features, the setting to require end-user input is "Require End-User Option Selection when Selection is Mandatory." For Connectors, the setting is "Require End-User Connection when Connection is Mandatory."

You can also require a variable to be bound by the end-user by creating a Configurator Extension that sets the variable's `InputRequiredFlag` System Property to `True` at runtime. For details about creating Configurator Extensions, see the *Oracle Configurator Developer User's Guide* and the *Oracle Configurator Extensions and Interface Object Developer's Guide*.

Note: If the end user invokes either the Finish or Auto-Complete action and there are one or more items that require end-user input, Auto-Complete does not run. In this case, a message displays all options that require end-user input before running Auto-Complete. Each option appears as a link that the end user can use to navigate to the page on which the option appears.

For more information, see:

- Input Required Message Box, page 3-54
- Input Required Dialog Page, page 3-54

Text Features

At runtime, only the end user can enter a value for a Text Feature. Therefore, Auto-Complete cannot complete a configuration if it contains a *required* Text Feature. A required Text Feature can be part of a completed configuration only if the end user has bound it by entering some text.

Note: Several types of nodes allow you to control whether they must be bound by the end-user at runtime. For details, see *Require End-User Input Setting*, page 3-19.

If your Model contains required Text Features and you use custom UI Content Templates, be sure to test the runtime Oracle Configurator thoroughly to be sure it performs as expected when one or more required Text Features is blank. (You can use the session-level System Property `InputRequired` to detect this case. For details, see the table *System Properties*, page 3-5.)

It is possible for a new instance of a component that contains a required Text Feature to be created when running Auto-Complete. When this happens, the status page that appears after Auto-Complete has finished contains a link that the end user can use to navigate to the page containing the Text Feature. For more information about the Auto-Complete process, see *Auto-Complete Configuration*, page 4-2.

At runtime, if an end user (or a Configurator Extension) attempts to set a value of a Text Feature that exceeds its defined Maximum Length, the resulting conflict cannot be overridden.

Counted Option Features

For an overview of Counted Option Features, see the *Oracle Configurator Developer User's Guide*.

In FCE Models, Counted Option Features include a setting that enables you to specify the maximum quantity an end user can enter for each option at runtime. This setting is called Maximum Quantity per Option. (The minimum quantity per option is implicitly set to 0 (zero), and you cannot change this value.)

Each Option within a Counted Option Feature has the following System Properties:

- `DefinitionMinQuantity`
- `DefinitionMaxQuantity`
- `MinQuantity`
- `MaxQuantity`
- `Quantity`

You can use the `QuantitySystemProperty` when defining rules.

Totals and Resources

In an FCE Model, you can create Totals and Resources of type Decimal or Integer. Integer Totals and Resources display only integers value at runtime, while Decimal Totals and Resources display either integer or decimal values. After creating a Total or Resource, you cannot change its type.

If you know that rules in your Model will be adding or subtracting only integer quantities to a Total or Resource, then use Integer Totals and Integer Resources. The integer type is preferable because Auto-Complete processes integer domains more efficiently at runtime.

Note: When you convert a Model to use the FCE, all existing Totals and Resources are converted to Decimal Totals and Decimal Resources. See *Converting Existing Models to use the Fusion Configurator Engine*, page 2-7.

When you create a Total or Resource, you must specify a Minimum and Maximum Value. By default, the predefined UI Content Templates that display Totals and Resources display their Minimum and Maximum Values at runtime (their domain). For example, a Decimal Total with a Minimum of 5.2. and a Maximum of 9.9 appears as follows when a configuration session begins:

Range: 5.2 to 9.9

Oracle Configurator dynamically updates the domain range as values are added and subtracted during a configuration session.

The range defined in Configurator Developer for both Integer and Decimal Resources is strictly enforced at runtime. For example, a Resource has Minimum and Maximum values of 1 and 10, respectively. At runtime, Oracle Configurator displays a conflict message if the end user's input or selection would cause the Resource to fall below 1 or exceed 10.

Initial Values

The FCE does not support the concept of initial values for Numeric Features, Totals, Resources, or Boolean Features in the same way as the Original Configurator Engine. Therefore, when you are editing an FCE Model, Oracle Configurator Developer does not provide a setting that enables you to specify an initial value for these node types.

If you want a Numeric Feature, Total, or Resource to have a specified value when a configuration session begins, then you must define a Rule that sets that value. You can define an Accumulator Rule to set the value, or an equivalent Statement Rule (using the `AddsTo` or `SubtractsFrom` operators). For example, to set the value of an Decimal Feature named `DecimalFeatureX` to 13.5 define a Statement Rule similar to the following:

Example

```
ADD 13.5 TO 'DecimalFeatureX'
```

Note that you can also set the value of a Numeric Feature from the value of some other Numeric Feature, as shown by the following:

Example

```
ADD 'DecimalFeatureY' TO 'DecimalFeatureX'
```

If you want a Boolean Feature to have a specified value when a configuration session begins, define a Statement Rule similar to the following and specify a Rule Class of 'Default':

Example

```
'BooleanFeatureY' = True
```

In FCE Models, it may be more accurate to think of an initial value as a *base* value, since the value you specify in Configurator Developer is not necessarily the value the node will have when the configuration begins. This is because it is possible for the node to participate in other rules which, when they propagate at runtime, modify the value that you specified for the node in your rules.

For example, you define a rule that sets DecimalFeatureX to 13.5 (as shown above), but an Option OptionA is selected by default and it participates in a rule that causes 10 to be added to DecimalFeatureX. In this case, the input range of DecimalFeatureX appears as shown below when the configuration session begins:

Range: 13.5 to 23.5

Note: When you convert a Model to use the FCE, all Initial Value settings for Numeric Features, Totals, and Resources are converted to Accumulator Rules. For example, if the Initial Value for IntegerFeatureX is 5, then after the conversion the FCE Model contains the following rule (shown here in Statement Rule format):

```
ADD 5 TO 'IntegerFeatureX'
```

Initial values on Boolean Features act as defaults. Therefore, they are converted to Implies Logic Rules.

BOM Nodes

This section describes some unique characteristics and runtime behaviors of BOM nodes in an FCE Model.

- In FCE Models, the System Property `RelativeQuantity` is available on all BOM nodes and can be used, for example, when defining rules or runtime UI conditions. At runtime, the value of this System Property is the current quantity of the node relative to its parent's quantity. For example, if a quantity of 4 is required for the child node when the parent node's quantity is 1, then the child's Relative Quantity is 16 when the parent's quantity is 4.

- When you import or refresh a BOM Model, or convert a Model to use the FCE, a default value is provided for any BOM nodes that have a blank (null) Minimum or Maximum Quantity. For details, see:
 - CZ: Default Max Quantity Integer, page 2-3
 - CZ: Default Max Quantity Decimal, page 2-3

Tip: For optimal performance of the Auto-Complete process, Oracle strongly recommends setting the Maximum Quantity as low as possible and the Minimum Quantity as high as possible when creating a BOM in Oracle Bills of Material. Oracle also recommends setting the Maximum Instances as low as possible and the Minimum Instances as high as possible for all BOM Model References in Configurator Developer.

BOM Model References

The following points are specific to BOM Model References.

- When configuring an FCE Model, Oracle Configurator automatically adds each instance of a BOM Model to the configuration as soon as the end user creates it. (When configuring a Model that uses the Original Configurator Engine, a newly created instance of a BOM Model is not added to the configuration until the end user *selects* it.)
- The maximum quantity allowed at runtime for an instance of a BOM Model is the Maximum Quantity defined for that item in Oracle Bills of Material. If you modify the Instances setting for a BOM Model Reference in Configurator Developer by selecting 'Multiple or Variable Instances', then the BOM Model's Maximum Quantity becomes the maximum quantity allowed for all instances of that item at runtime.

When viewing a BOM Model Reference's details page in Configurator Developer, the Initial Minimum Instances and Initial Maximum Instances settings are not displayed if the node can have multiple instances at runtime (that is, when 'Multiple or Variable Instances' is selected). In this case, Configurator Developer displays text similar to the following in the node's details page:

Note: The Maximum Quantity defines the total Quantity allowed across all instances of this component.

- You can specify whether a BOM Model Reference that is a child of a BOM Option Class is required or optional at runtime. In other words, you can select either 'Required Single Instance' or 'Optional Single Instance' in the node's details page in Configurator Developer. Because a BOM Model Reference that is a child of a BOM Option Class cannot be instantiable multiple times, the 'Multiple or Variable

Instances' setting is disabled for such nodes in Configurator Developer.

Connectors

Important: Connectors are not supported in the current release of the Fusion Configurator Engine. The information about Connectors provided in this document is only for planning purposes. However, Connectors continue to be supported for models using the Original Configurator Engine.

In an FCE Model, you can define single-instance Connectors, and also Connectors with Multiple-Instance and and Reverse relationships. This section describes Multiple-Instance and Reverse Connectors. For details about single-instance Connectors, and general information about Connectors, see the *Oracle Configurator Developer User's Guide*.

Important: To create any type of Connector, both the source and the target Model must use the FCE. Therefore, when editing an FCE Model, only FCE Models are displayed when selecting the Connector's target Model.

Multiple-Instance Connectors

A Multiple-Instance Connector is a Connector that can have multiple targets at runtime, and therefore may be connected multiple times within a configuration. For example, to configure a local area network (LAN), an end user must be able to connect multiple devices - such as printers and workstation - to a single server hub.

You define a Multiple-Instance Connector in Configurator Developer by specifying values for the Minimum and Maximum Connections settings. These settings appear in a Connector's details page, and the default values are 0 and 1, respectively.

Note: When you convert a Model to use the FCE, the default Minimum and Maximum Connections for all required Connectors is 1/1. (A required Connector is one whose Connection Required setting is selected in the source Model.) Additionally, the Connection Required setting (Connector details page) is not available in FCE Models. For an example of how you can make a connection required at runtime in an FCE Model, see *Connectors and Configuration Rules*, page 3-26.

Oracle Configurator Developer uses the Multiple-Instance Connector Control UI Content Template to display Multiple-Instance Connectors at runtime. For details, see *Multiple-Instance Connector Control*, page 3-62.

Reverse Connectors

A Reverse Connector defines a reverse relationship with a Connector in the Connector's target Model. When this relationship exists, creating a connection at runtime creates a bi-directional connection between the Connector and its target Model. In other words, when the end user creates a connection or disconnects an existing connection, the other Connector in the reverse relationship is connected or disconnected automatically.

Reverse Connectors are useful when, for example, you need to define rules that are based on a connection occurring at runtime, regardless of whether the connection is made in the source or the target Model. When a Model contains a Reverse Connector, any rules that you define in the target Model can include nodes from the Connector's parent Model as participants (the reverse is also true). If a Reverse Connector also allows multiple connections, it is not necessary to duplicate the same rule for every possible connection source.

You can modify the Reverse Connector setting in either the source or target Model in Configurator Developer, as long as the other Model is not locked by another user. When you remove or modify a reverse connector relationship in one Model, Oracle configurator makes the corresponding change in the other Model.

When you perform any of the following actions on a Reverse Connector, Configurator Developer prompts you to confirm the action and then (if you choose to proceed) clears the reverse relationship:

- Delete the Connector
- Move the Connector so it is no longer a child of the root Model
- Change the Connector's target (that is, select a different Model)
- Copy the Connector: In this case, Configurator Developer clears the reverse relationship in the new (copied) Connector, but the relationship in the source Connector does not change.

At runtime, an end user can connect or disconnect either participant in a reverse relationship. When this happens, the other participating component also connects or disconnects automatically.

Modifying or removing a reverse connection relationship does not affect any existing rules in which the Connectors participate.

Procedure

To create a Reverse or Multiple-Instance Connector:

1. Open a Model for editing, and then create a Connector.

See the *Oracle Configurator Developer User's Guide* for details.

2. In the Connector's details page, specify values for **Minimum Connections** and

Maximum Connections. The default values are 0/1. Accept these values if you do not want to allow multiple connections at runtime.

3. If you want the Connector to participate in a reverse relationship, select a Connector from the **Reverse Connector** list.

Note: If the Connector you are viewing is not a child of the root Model, the Reverse Connector setting is disabled.

This list displays all Connectors in the target Model that:

- Do not already participate in a reverse relationship with another Connector
- Are children of the root Model

In other words, Connectors that are children of a Model Reference or a Component do not appear in the list.

4. Specify Effectivity settings, and then click **Apply**.

Important: To create any type of Connector, both the source and the target Model must use the FCE. When selecting a Connector from the Reverse Connector list, Configurator Developer displays only Connectors that exist within FCE Models.

Connectors and Configuration Rules

To make a connection required at runtime, define a Statement Rule that uses the Connector and its `ConnectionCount` System Property. For example:

```
FeatureA IMPLIES ConnectorA.ConnectionCount() >= 1
```

To prevent a connection from being made at runtime, define a Statement Rule similar to the following:

```
FeatureA IMPLIES ConnectorA.ConnectionCount() = 0
```

For additional examples of rules involving the `ConnectionCount` System Property, see Model Node System Properties, page 3-3.

Effectivity

In an FCE Model, Effectivity settings are available for the following node types:

- BOM nodes (all types)
- Option Features
- Options

- Boolean Features
- Optional and instantiable Components

Note: After converting an existing Model to use the FCE, Effectivity settings are not available for Integer and Decimal Features, Totals, Resources, and mandatory Components (that is, Components with a Min/Max of 1/1). To work around this change, use Display Conditions to control the display of these nodes at runtime.

Additionally, you can specify Effectivity for any type of rule in an FCE Model (including Accumulator Rules).

Configuration Rules

This section describes unique characteristics of configuration rules in FCE Models. For general information about configuration rules, see the *Oracle Configurator Developer User's Guide*.

Logic States

In an FCE Model, a variable that has neither been selected nor excluded from a configuration at runtime has a logic state of Unbound. (In Models that use the Original Configurator Engine, options meeting this criteria have a logic state of Unknown.) A variable is unbound when its domain is open, which means that either a value has not been assigned or the set of its members has not been finalized. Variables may be unbound because the end user has not yet made a selection, entered a value, or run Auto-Complete.

At runtime, the value of the `SelectionMode` and `DetailedSelectionMode` System Properties is `Selectable` for options that are neither selected nor excluded from a configuration. For information about how logic state determines an option's appearance at runtime, see Logic State Display, page 4-1.

Note: When you convert a Model to the FCE, the Unknown logic state becomes Unbound, Not Selected becomes Selectable, and Unsatisfied becomes Input Required.

Refer to the *Oracle Configurator Developer User's Guide* for details about other logic states.

Rule Classes

When defining a rule in Configurator Developer, you must assign the rule to a Rule Class. Oracle Configurator refers to a rule's Rule Class when an end user is manually configuring a product and when the Auto-Complete process is running.

A rule's Rule Class determines the following at runtime:

- The rule's general behavior
- Whether the rule is mandatory (that is, it must always be True in the configuration)
- At what point in the configuration session the rule is applied (Defaults and Search Decisions only)

The three Rule Classes are:

- Constraint, page 3-29
- Default, page 3-30
- Search Decision, page 3-31

Which Rule Class you can specify depends on the rule's type. Configurator Extensions and Rule Sequences cannot have a Rule Class. All other types of rules must have a Rule Class, but not all Rule Classes are valid for each type of rule. For details, see the table Rule Types and Rule Classes, page 3-29. If multiple Rule Classes are available for a rule, you can change its Rule Class at any time.

Caution: New Defaults and Search Decisions appear at the end of their respective sequence, and changing an existing rule's Rule Class from Default or Search Decision to Constraint may adversely affect a well-defined sequence. Therefore, be sure to review and unit test the sequence of Defaults and Search Decisions after modifying a rule's Rule Class. See Specifying a Sequence for Defaults and Search Decisions, page 3-31.

Rule Sequences can contain any rule, regardless of the rule's class. However, to reduce future maintenance when upgrading to a future version of Oracle Configurator, Oracle strongly recommends that all rules in each Rule Sequences have the same Rule Class. For example, define a Rule Sequence that consists of only Constraints. For details about Rule Sequences, see the *Oracle Configurator Developer User's Guide*.

Certain Constraint Definition Language (CDL) operators are not available for use in Statement Rules, depending on a rule's Rule Class. For example, a Statement Rule using an Accumulator operator (AddsTo or SubtractsFrom) can only have a Rule Class of Constraint. For details, see Fusion Configurator Engine and the Constraint Definition Language, page 3-37.

Rule Types and Rule Classes

Rule Type	Constraint	Default	Search Decision
Logic Rule	X	X	X
Comparison Rule	X	X	X
Accumulator Rule	X		
Property-based Compatibility	X		
Explicit Compatibility	X		
Design Chart	X		
Statement Rule	X	X	X

Constraints

Constraints are applied at runtime while an end user manually selects options and enters values during a configuration session.

Rules that are classified as Constraints are applied at runtime while an end user manually selects options and enters values during a configuration session. Constraints must *always* be true in the context of a configuration. For example, when the end user makes a selection that violates a Constraint at runtime, Oracle Configurator displays a contradiction message informing the user that the previous action cannot be applied.

A Constraint may be expressed as a logical expression, a numeric comparison, a compatibility table (or Design Chart), or a property-based compatibility expression.

Unlike Defaults and Search Decisions, you cannot specify the order in which you want Oracle Configurator to consider Constraints at runtime.

For a list of which Rule Classes are valid for each type of rule, see the table Rule Types and Rule Classes, page 3-29.

In an FCE Model, relational operators can be the primary operator within a Constraint. Consider the following examples that use the equals (=) and greater-than (>) relational operators:

$x = y + (q * z)$

$a > b$

In these simplified expressions of rules, the left-hand side of the expression can propagate (or "push") numeric information to the right-hand side. In the FCE, the right-hand side of the expression can *also* propagate ("push back") to the left-hand side. The ability to define such Constraints allows rules in FCE Models to be bidirectional; that is, they can propagate in both directions.

Defaults

Like Constraints, Defaults are also applied at runtime while an end user manually selects options and enters values during a configuration session. However, unlike Constraints, Defaults are:

- Flexible, and they do not lead to a contradiction at runtime when, for example, the end user deselects an option that was selected by the rule.
- Applied at runtime in the order specified in Configurator Developer, after each of the end user's inputs are applied and propagated.

A Default can fail due to a conflict with one of the end user's inputs or the propagation of a Constraint. (You can specify the order in which Defaults propagate; see *Specifying a Sequence for Defaults and Search Decisions*, page 3-31.)

You can use Defaults to guide end users towards a preferred solution by defining several contradictory rules that will be processed in the order you specify at runtime.

For example, a manufacturer of laptop computers prefers that their customers purchase the lightweight version of a laptop instead of the heavier model, and the Deluxe carrying case rather than the Basic version. To guide buyers towards purchasing the lightweight laptop with the Deluxe case, without preventing them from selecting alternative options, the manufacturer defines the following rules and sequence:

1. Laptop 900 Implies 900-LTW
2. Laptop 900 Implies Deluxe Case
3. Laptop 900-HVW Implies Deluxe Case

With these rules in place, the lightweight version of the laptop (900-LTW) and the Deluxe Case will be selected by default when the end user selects the Laptop 900 model. If the end user then selects the heavier model (the 900-HVW), the Deluxe Case will still be selected.

Many constraints can be defined as a Default. For example, your Model contains a Numeric Feature called Weight which has a range (domain) of 1000 - 5000. You prefer a solution in which the value of this item is less than or equal to 3000, so you defined the following Statement Rule and assign a Rule Class of Default:

```
Weight <= 3000
```

When this rule propagates at runtime, the range for the Weight item is reduced and appears as follows in the runtime UI:

```
Range: 1000 to 3000
```

For a list of which types of rules can be classified as Defaults, see *Rule Types and Rule Classes*, page 3-29.

Note: After converting an existing Model to use the FCE, the Defaults operator does not appear in the Logic Rule or Comparison Rule details pages.

Search Decisions

Rules that you classify as Search Decisions are applied:

- During the Auto-Complete process
- After all User Decisions and Defaults have been applied and propagated
- Before the application of the FCE's inherent Search Decisions
- In the order that you specify in Configurator Developer

Rules classified as Search Decisions may be expressed as a logical expression or a numeric comparison.

For more information, see *Specifying a Sequence for Defaults and Search Decisions*, page 3-31.

For a list of which Rule Classes are valid for each type of rule, see *Rule Types and Rule Classes*, page 3-29.

Specifying a Sequence for Defaults and Search Decisions

At runtime, Auto-Complete applies rules classified as Defaults and Search Decisions according to the sequence that you specify in Configurator Developer. When you define a rule and classify it as either a Default or Search Decision, Configurator Developer assigns a default sequence number which places the rule at the end of its respective list. For example, you have five existing rules with the "Defaults" Rule Class. When you create a new rule and specify a Rule Class of Defaults, the new rule appears at the end of the list of Defaults rules, and its sequence number is 6.

If you want a rule to appear earlier in its respective sequence, click either *Reorder Defaults* or *Reorder Search Decisions* in the Rules area of the Workbench.

Note: At runtime, any rules that have cross-model or cross-instance participants will not necessarily be applied in their specified sequence relative to the other rules defined in the Model. The order of a set of such rules that apply to the same scope (combination of Models or instances) will remain as defined relative to one another, but as a group they will be applied after instantiation of the scope (all the rule's participants) and application of all the rules of that class defined within the various instances of the scope.

Procedure

To modify the order in which Defaults or Search Decisions are applied at runtime:

1. From the Rules area of the Workbench, click **Reorder Defaults** or **Reorder Search Decisions**.
2. Specify a new sequence number for one or more rules, and then click **Update**.
3. Review the updated sequence. The Changed column indicates which rules you updated.
4. If you are satisfied with the changes, click **Apply**.

To cancel your changes and return to the Rules area of the Workbench, click **Cancel**.

Compatibility Rules

This section describes characteristics of Compatibility Rules that apply only in FCE Models.

When defining any type of Compatibility Rule (Explicit, Property-based, or Design Chart), Configurator Developer does not allow more than one of the rule's participants to have a Maximum Selections value that is greater than 1. If you violate this restriction when creating or editing a Compatibility Rule, Configurator Developer displays a warning when you save or validate the rule. If the Maximum Selections value of one of the rule participant's changes after the rule is created, Configurator Developer displays an error when you generate logic.

Enforcing this restriction in Configurator Developer ensures that Compatibility Rules perform as expected at runtime.

Accumulator Rules

Accumulator Rules allow you to add or subtract a value from a variable at runtime. For example, when the end user selects the 512 MB RAM option you want to add 512 to a Total called "Total RAM Selected."

Procedure

You can create an Accumulator Rule by either of these methods:

- Graphically:
 1. Create a Rule of type Accumulator Rule.
 2. Specify which Model nodes are participants (operands) in the rule.
 3. Select either the AddsTo or SubtractsFrom operator.
- Programmatically:

1. Create a Rule of type Statement Rule.
2. Enter the text of the Statement Rule, using the AddsTo or SubtractsFrom operator according to the following syntax:

`ADD n TO x`

or

`SUBTRACT n FROM x`

Where *n* is a number and *x* is a node, such as a Total, a Resource, or a Numeric Feature.

Accumulator Rules can only have a Rule Class of Constraint; they cannot be classified as Defaults or Search Decisions. If you create a Statement Rule with a Rule Class of either Defaults or Search Decision, and the rule's text defines an Accumulator rule (that is, it uses either the "AddsTo" or "SubtractsFrom" operators), then Configurator Developer forces the Rule Class setting to Constraints, and displays a message similar to the following when you validate the rule:

Rule Class changed to 'Constraint'.

If you later change the text of the Statement Rule such that it no longer defines an Accumulator Rule, then Configurator Developer re-enables the Rule Class setting. For more information about Rule Classes, see Rule Classes, page 3-27.

Note: When you convert an existing Model to use the FCE, all Contributes and Consumes Rules are converted into Accumulator Rules, and "Numeric Rule" is no longer an option when creating new rules in the converted Model. Additionally, any Initial Value that is defined for a Numeric Feature, Total, or Resource is removed and converted into an Accumulator Rule. For more information, see Converting Existing Models to Use the Fusion Configurator Engine, page 2-7.

It also is important to understand that Accumulator Rules do not simply add or subtract a quantity from a variable. All rules of this type defined against the same target node can be considered terms in a constraint against that node. This is because all AddsTo and SubtractsFrom expressions in a Model become a single constraint on the target node. In other words, the target node equals the sum of all AddsTo expressions defined against it in the Model minus the sum of the SubtractFrom expressions.

Additionally, if the target node is involved in any other constraints, the equality constraint generated by its AddsTo and SubtractsFrom expressions must be satisfied along with all the others. As with all other constraints, the equality constraint is bidirectional, so it can "push back" on the values of the participants on the left-hand-side of the rule.

Keep the following in mind when using Accumulator Rules:

- If the Model contains multiple Accumulator Rules that add to or subtract from the same target node, and that node exists in a *referenced* Model, generating logic creates a single constraint that equates the target to a sum of all the terms expressed in the individual rules in that model.

If AddsTo or SubtractsFrom rules are defined against a given target within multiple parent models in a reference model hierarchy, each of the generated equality constraints must be satisfied individually. In other words, the AddsTo and SubtractsFrom terms are not accumulated across multiple referencing models.

- Since the FCE merges all of a Model's Accumulator Rules into a single Constraint, it is not possible to support individual rule violation messages for Accumulator Rules. Consequently, Configurator Developer provides no Rule Violation section in an Accumulator Rule's details page. Similarly, the Rule Violation section is disabled for Statement Rules when their text defines an Accumulator Rule (that is, uses the AddsTo or SubtractsFrom operator). If a Statement Rule's text changes such that it no longer defines an Accumulator Rule, then Configurator Developer re-enables the Rule Violation section.
- When you generate Model logic, Configurator Developer displays a warning message for any nodes that are the target of Accumulator Rules from multiple Models. The message warns that multiple (and possibly contradictory) rules were generated for the target node, and that these rules may produce undesirable results at runtime. In this case, it may be necessary to modify the constraints defined in the Model containing the target node.

Note: When you convert an existing Model to use the FCE and one or more Numeric Rules meet the criteria described in the preceding paragraph, similar warnings appear in the Model Conversion Report. For more information, see *Converting Existing Models to Use the Fusion Configurator Engine*, page 2-7.

Configurator Extensions

This section describes information about Configurator Extensions (CX) that you should consider when using the FCE.

In a Model that uses the FCE, the Configurator Extension event `postValueChange` can be triggered only when the variable (option) to which it is associated is bound at runtime. (For example, a Numeric Feature is bound when it has a value.) This event cannot be triggered if the variable's domain changes, but the variable itself remains unbound.

Configurator Extension events cannot be triggered during the Auto-Complete process. If a CX is used to modify any part of a completed configuration that was created by Auto-Complete (including states, values, instance containments, and so on), then Oracle

Configurator displays an error. There is one exception: a CX can be used to modify a configuration that was created by Auto-Complete after the end user returns to the configuration to make changes. To determine whether a configuration was created by Auto-Complete, write your CX such that it queries the session property `ConfigComplete()`. To determine whether the end user is making changes to a configuration that was created by Auto-Complete, write your CX such that it queries the session property `InAdjustMode()`.

The table *Predefined Events for Binding*, page 3-35 describes the CX events that are available only in an FCE Model. The table also lists the event-specific parameters that you use as arguments when binding the method parameters of a Java class in a Configurator Extension Rule.

Predefined Events for Binding

Event Name	Related To	Description	Event Parameter Type	Event Binding Scope
<code>preAutoComplete</code>	Configurator Extension	Event dispatched just before the Auto-Complete process is initiated, either on request or implicitly in a Finish flow.	None	Global Only

Event Name	Related To	Description	Event Parameter Type	Event Binding Scope
postAutoComplete	Configurator Extension	Event dispatched just after Auto-Complete terminates, whether it succeeds or not. The status of the operation and the state of the configuration can be tested using session properties such as <code>AutoCompleteSuccessful</code> and <code>ConfigComplete</code> etc.	None	Global Only

The following CX events are *not* supported in FCE Models:

- `postInstanceEditable`
- `postInstanceNonEditable`
- `onInstanceLoad`
- `postInstanceLoad`
- `onValidateEligibleTarget`
- `postConnect`
- `postDisconnect`
- `onConfigLineType`
- `onConfigValidate`

For a complete list of events that are available in Models that use either the original configuration or the FCE, see the *Oracle Configurator Developer User's Guide*.

Configurator Extensions and Converted Models

If you are converting existing Models to FCE Models, it is important to note that the FCE provides functionality that eliminates the need for some types of Configurator Extensions.

Additionally, any CXs that were created for a Model that used the Original Configurator Engine must be modified if you want to use them after converting the Model to use the FCE.

Fusion Configurator Engine and the Constraint Definition Language

For general information about the Constraint Definition Language (CDL), refer to the *Constraint Definition Language Guide*.

This section lists CDL functions and operators that are available only in FCE Models, as well as some obsolete functions and operators.

The following functions are not available in FCE Models:

- NOTTRUE function
- ATAN2 operator

The table Operators Listed by Type, page 3-38 describes operators and functions that are available only in FCE Models.

Operators Listed by Type

Operator Type	Operator	Description
Arithmetic function	AggregateSum (function)	<p>Can be used in a Constraint, Default, or Search Decision, but only as a sub-expression.</p> <p>Syntax:</p> <pre>AGGREGATESUM ([NodeRef NodePropRef])</pre> <p>where Noderef is a path-based reference to a node within each member instance of InstanceSet, and NodePropRef is a NodeRef with a numeric or logic property specified.</p> <p>The return value of the function is the sum of the values of NodeRef or NodePropRef for all the instances implied in the path.</p>
Other	Assign	<p>Used only in Defaults and Search Decisions to force a node to be bound at a particular point in the specified sequence.</p> <p>If the Domain Ordering setting is specified in the node's details page, binding occurs according to this setting. Otherwise, the FCE's implicit binding method for this operator type is used.</p> <p>Syntax:</p> <pre>ASSIGN (node)</pre>

Operator Type	Operator	Description
Other	IncMin	<p>Used only in Defaults and Search Decisions.</p> <p>Similar to ASSIGN, but this operator overrides any explicit or implicit domain ordering method for binding the node and attempts a binding using a binary search with increasing minimum.</p> <p>This operator is valid for integers and decimals, including BOM items and Options with quantity. Applies to the node's default System Property when a System Property is not explicitly referenced (for example, State, Quantity, or Value).</p> <p>When used with BOM items, you can specify the <code>RelativeQuantity</code> property as an alternative.</p> <p>Syntax:</p> <pre>INCMIN([node node.Quantity() numericnode.Value() bomnode.RelativeQuantity()])</pre>

Operator Type	Operator	Description
Other	DecMax	<p>Used only in Defaults and Search Decisions.</p> <p>Similar to ASSIGN, but this operator overrides any explicit or implicit domain ordering method for binding the node and attempts a binding using a binary search with decreasing maximum.</p> <p>This operator is valid for integers and decimals including BOM items and Options with quantity. Applies to the node's default System Property when a System Property is not explicitly referenced (for example, State, Quantity, or Value).</p> <p>When used with BOM items, you can specify the <code>RelativeQuantity</code> property as an alternative.</p> <p>Syntax:</p> <pre>DECMAX([node node.Quantity() numericnode.Value() bomnode.RelativeQuantity()])</pre>

Operator Type	Operator	Description
Other	MinFirst	<p>Similar to ASSIGN, but this operator temporarily overrides any explicit or implicit domain ordering method for the node and attempts a binding using a linear search beginning with the node's specified minimum value.</p> <p>Used only in Defaults and Search Decisions.</p> <p>Valid for integers only.</p> <p>Syntax:</p> <ul style="list-style-type: none"> • <code>MINFIRST (node)</code>
Other	MaxFirst	<p>Similar to ASSIGN, but this operator temporarily overrides any explicit or implicit domain ordering method for the node and attempts a binding using a linear search beginning with the node's specified maximum value.</p> <p>Used only in Defaults and Search Decisions.</p> <p>Valid for integers only.</p> <p>Syntax:</p> <ul style="list-style-type: none"> • <code>MAXFIRST (node)</code>

Operator Type	Operator	Description
Arithmetic	Add ... To	<p>Specifies addition of a numeric value to a Total, Resource, Numeric Feature, Option quantity, ...</p> <p>or the minimum or maximum number of component instances that are allowed at runtime.</p> <p>Calculation of the numeric value can involve Constants, Boolean values, Numeric Features, Option quantities, mutable System Properties, Totals, ...</p> <p>the minimum and maximum number of instances, and the instance count.</p> <p>Syntax:</p> <pre>ADD n TO x</pre>
Arithmetic	Subtract ... From	<p>Specifies subtraction from a numeric value to a Total, Resource, Numeric Feature, Option quantity, ...</p> <p>or the minimum or maximum number of component instances that are allowed at runtime.</p> <p>Calculation of the numeric value can involve the same nodes and objects listed in "Add ... To", above.</p> <p>Equivalent to 'ADD -(n) TO x'</p> <p>Syntax:</p> <pre>SUBTRACT x FROM y</pre>

Operator Type	Operator	Description
Logical	SubSetOf	<p>Can be used in a Constraint, Default Decision, or Search Decision as a conditional expression or a top-level constraint.</p> <p>Returns Boolean True or False. In the FCE, this operator is only supported between two ports (Connectors, InstanceSets, or computed ports).</p> <p>Syntax:</p> <pre>port1 SUBSETOF port2</pre> <p>For more information, seeSubSetOf and Union Operators, page 3-44.</p>
Other? Logical?	Union	<p>Can be used in a Constraint, Default Decision, or Search Decision, but only as a sub-expression.</p> <p>Returns the union of the members of all specified ports. The return value can be used as an argument of type <i>port</i> in an enclosing expression.</p> <p>Syntax:</p> <pre>UNION(port1, port2 ..., portn)</pre> <p>For more information, seeSubSetOf and Union Operators, page 3-44.</p>

SubSetOf and Union Operators

SubSetOf Operator

For an overview of the SubSetOf operator, see the table Operators Listed by Type, page 3-38.

Consider an expression of the form:

```
port1 SUBSETOF port2
```

When such an expression using the SubSetOf operator returns True, the expression constrains the members of *port1* to be a subset of the members of *port2*. (An identical set also qualifies). If the port is an InstanceSet, the members are the contained Component Instances or Model Instances. If the port is a Connector, the members are the connected Model Instances.

For example, when the following expression is True, it states that the target of Connector-1 must be one of the instances belonging to InstanceSet-1:

```
'Connector-1 SUBSETOF InstanceSet-1'
```

Membership in an InstanceSet is exclusive, therefore the following expression will prevent any instances contained in InstanceSet-2 from occurring in InstanceSet-1:

```
'InstanceSet-1 SUBSETOF InstanceSet-2'
```

However, the following expression will succeed if all the target instances of Connector-1 are also assigned to Connector-2:

```
'Connector-1 SUBSETOF Connector-2'
```

Note: A port is a variable whose domain is a set of Models that is contained by or associated with the Model where the port is declared. The FCE supports two types of port variables, which can connect to existing Model instances or generate new ones as needed. One type is an InstanceSet variable, which represents a composition relationship to sub-components. The other type is a ConnectorSet variable, which represents an association relationship to peer Model instances. Therefore, members of InstanceSets are mutually exclusive, while ConnectorSets do not have such restriction. In other words, an instance can be a member of multiple ConnectorSets, but it can be a member of only one InstanceSet.

UNION Operator

For an overview of the Union operator, see the table Operators Listed by Type, page 3-38.

When using this operator, the port operands can be InstanceSets, Connectors, or a combination of the two, as long as they both have the same target type – that is, the target model of a Connector or of an instantiable Model Reference, or the Component identity of an instantiable Component.

An Instance Set representing an instantiable Component (as opposed to a Model Reference) can only participate in a UNION when all of the other operands are the same instantiable Component in a different instance of the parent Model.

For example, your Model has the following structure:


```

M1
|--C1 (0,n)
|->M2 (0,n)
|   |--C2 (0,n)
|->M3 (1, 1)
|   |--C3 (0,n)
|->M3' (1, 1)
|   |--C3 (0, n)

```

Note: Models M2, M3, and M3' are referenced by Model M1.

In this example, the UNION operation is valid on M3.C3 and M3'.C3, since they have the same Component identity in two different (fixed) instances of M3. The UNION operation would also be valid on M2(m).C2 and M2(n).C2, if it were possible to refer to specific instances of M2 when defining the Model's structure in Configurator Developer, but this is not the case. Therefore, no other Component in Model M1 is a valid participant in a UNION.

Comparison Operators

In a Model that uses the Original Configurator Engine, comparison operators such as equals, not equals, greater than, less than, and so on can be used only in conditional sub-expressions. For example:

```
IF (x < y) THEN ...
```

In an FCE Model, you can also use comparison operators as "top-level" constraints, which means they do not have to be used in a sub-expression. For example, the following expression can constitute an entire rule definition, constraining *x* to be less than *y*:

Example

```
x < y
```

Rule Import for Fusion Configurator Engine Models

Rule import, which is the ability to import configuration rules in CDL format into the CZ schema, applies to FCE Models as well as to Original Configurator Engine (OCE) Models.

The general procedure for rule import is the same for FCE Models as for OCE Models. The procedure for OCE Models is described in the section on Rule Import in the *Oracle Configurator Implementation Guide*. The only difference for FCE Models is that you must populate several columns in the table CZ_IMP_RULES *in addition to* the columns listed in the Implementation Guide. The additional columns required for FCE rule import are as follows:

- CLASS_SEQ (nullable, type NUMBER): The sequence number for the rule within its current Rule Class. Values for this number should not have any decimal part. For background, see Specifying a Sequence for Defaults and Search Decisions, page 3-31

Within each Rule Class (RULE_CLASS) for the specified Model (DEVL_PROJECT_ID), each rule must have a unique class sequence number (CLASS_SEQ).

A value should be provided only for FCE Models. A value is mandatory if CONFIG_ENGINE_TYPE is 'F' and RULE_CLASS is either 1 or 2. The value should be null if RULE_CLASS is 0 or null.

- CONFIG_ENGINE_TYPE (nullable, type VARCHAR2(1)): The Configurator Engine Type of the Model into which you are importing rules.

The valid values are:

F - Fusion (FCE)

L - Original (OCE/LCE)

The value must be 'F' for FCE Models.

- RULE_CLASS (nullable, type NUMBER): The numeric identifier of the Rule Class of the rule. For background, see Rule Classes, page 3-27].

The valid values are:

0 - Constraint

1 - Default

2 - Search Decision

A value should be provided only for FCE Models. A value is mandatory if CONFIG_ENGINE_TYPE is 'F'. The value should be null if CONFIG_ENGINE_TYPE is 'L'.

For details on CZ_IMP_RULES and other Oracle Configurator tables, see the CZ *e*TRM on My Oracle Support.

Note: The CZ schema for this release also includes two nullable columns that are not used for rule import: ACCUMULATOR_FLAG and TOP_LEVEL_CONSTRAINT_FLAG. Do not populate these columns, and ignore any values that might appear in them after the rule import process.

Validation Messages for FCE Rule Import

The messages related to rule import for the FCE are described in the table Validation Messages for FCE Rule Import, page 3-47, along with the error or warning conditions that can trigger the messages. Some messages can be triggered by more than one condition. These messages appear after the import in CZ_IMP_RULES.MESSAGE.

Validation Messages for FCE Rule Import

Message	Error or Warning Conditions
Invalid Class Sequence	<ul style="list-style-type: none">• CLASS_SEQ is provided when RULE_CLASS is 0 (Constraint).• CLASS_SEQ is not provided when RULE_CLASS is 1 (Default) or 2 (Search Decision).• CLASS_SEQ is not unique for the specified Model and RULE_CLASS.
Invalid Configurator Engine Type	<ul style="list-style-type: none">• CONFIG_ENGINE_TYPE is not 'F' or 'L'.
Combination of Configurator engine type, Rule Class and Class Sequence is invalid	<ul style="list-style-type: none">• CLASS_SEQ or RULE_CLASS are provided when CONFIG_ENGINE_TYPE is 'L'.
Invalid Rule Class	<ul style="list-style-type: none">• RULE_CLASS is not provided when CONFIG_ENGINE_TYPE is 'F'.• RULE_CLASS is not 0, 1, or 2.
Custom Rule Violation Messages are not supported for Accumulator Rules	<ul style="list-style-type: none">• CONFIG_ENGINE_TYPE is 'F' <i>and</i> the rule is an Accumulator Rule <i>and</i> RULE_CLASS is not 0 <i>and</i> CZ_IMP_RULES.REASON_TYPE is not 0 (rule name).• Note: Custom Rule Violation Messages are stored in CZ_LOCALIZED_TEXTS.LOCALIZED_STR.• For background on why Accumulator Rules do not support rule violation messages, see Accumulator Rules, page 3-32.

Message	Error or Warning Conditions
Rule Class for Accumulator and Compatibility statements should be Constraint	<ul style="list-style-type: none"> The statement rule definition in RULE_TEXT indicates that the imported rule is an Accumulator or Compatibility rule, and RULE_CLASS is not 0 (Constraint).
Operator &OP requires Rule Class of either Default or Search Decision.	<ul style="list-style-type: none"> RULE_CLASS is 0 (Constraint) and the statement rule definition in RULE_TEXT includes one of the operators ASSIGN, INCMIN, DECMAX, MINFIRST, or MAXFIRST. Note: The message token &OP is replaced by the operator detected in the import record.

Creating and Editing a User Interface

This section describes the UI Content Templates that are available only when editing an FCE Model, and additional content that is available in various UI templates when the FCE is enabled. For more information about creating and editing User Interfaces, see the *Oracle Configurator Developer User's Guide*.

User Interface Master Templates

When the FCE is enabled, by itself or in addition to the OCE, UI Master Templates include settings and content that is specific to the FCE. For details about enabling the FCE, see CZ: Enable Configurator Engine, page 2-3.

Following is a complete list of settings and content that appears in a UI Master Template when the FCE is enabled:

- **Pagination and Layout:** This section contains the Enable Auto-Override for Conflicts setting, and this setting is selected by default.

For more information, see Auto-Override, page 4-10.

- **Non-BOM Content Custom Settings:** This page contains separate settings for specifying which templates to use when displaying the following content at runtime:
 - Integer Features (default is Integer Input with Range Display)
 - Decimal Features (default is Decimal Input with Range Display)

- Single-Instance Connectors (default is Connection Control)
- Multiple-Instance Connectors (default is Connection Management Table)
- **Utility Templates:** This section contains the following settings:
 - Instance Chooser (default is Instance Chooser Page)
 - Auto-Complete Status (default is Auto-Complete Status Dialog)
- **Required Messages:** This section contains settings for specifying which UI Content templates to use when displaying required messages at runtime. Separate sections are provided for Models that use the Original Configurator Engine and Models that use the Fusion Configurator Engine.
- **Optional Messages:** This section contains settings for specifying which UI Content templates to use when displaying optional messages at runtime. The Undo Status setting is relevant only for FCE Models.
- **Images:** Following is a summary of how this section changes when the FCE is enabled (except where indicated):
 - The Not Selected setting changes to **Selectable** (this is true whether or not the FCE is enabled).
 - You can specify which images are used to display Enhanced Check Boxes and Enhanced Radio Buttons with a status of **Recommended (Proposed Selected)** and **Not Recommended (Proposed Excluded)**.
 - In the **Status Indicator Images** region, you can specify which images are used to indicate options with a status of Recommended or Not Recommended.
 - In the **Status Indicator Images** region, you can specify which images are used to indicate options with a status of Proposed, Changed, Input Required, and Input Error.

For more information about the images that are used by default, see Runtime Icons and Images, page 3-66.

For details about the UI Content Templates that are used by default, see User Interface Content Templates, page 3-49.

User Interface Content Templates

When the FCE is enabled, by itself or in addition to the OCE, additional UI Content Templates are available for use. For details about enabling the FCE, see CZ: Enable Configurator Engine, page 2-3.

This section includes the following sections:

- Control Templates, page 3-50
- Utility Templates, page 3-52
- Message Templates, page 3-54
- Button Bar Templates, page 3-56
- Other Templates, page 3-59
- Changes to Existing User Interface Content Templates, page 3-60

Control Templates

The UI Content Templates in this section are located in the Control Templates folder in the Main area of the Repository. They are also available when you are working in a custom UI Master Template and are selecting templates in either the BOM Content Custom Settings page or the Non-BOM Content Custom Settings page.

Integer Input with Range Display

This template displays an input field for an Integer Feature at runtime, with text indicating the range of values that the field will accept. For example, if an Integer Feature's Minimum is set to 1 and its Maximum is set to 9 in Configurator Developer, then the text below the input field appears as follows:

Range: 1 to 9

When the Feature is bound (by Auto-Complete, for example), the text no longer appears.

The predefined Integer Input with Range Display template is located in the Value Display & Input subfolder in the Control Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template when defining custom settings for displaying non-BOM content, and it is the default for the Integer Features setting.

Decimal Input with Range Display

This template displays an input field for a Decimal Feature at runtime, with text indicating the range of values that the field will accept (the Feature's domain). For example, if a Decimal Feature's Minimum is set to 1.5 and its Maximum is set to 4.9 in Configurator Developer, then the text below the Feature's input field appears as follows:

Range: 1.5 to 4.9

When the Feature is bound (by Auto-Complete, for example), the text no longer appears.

The predefined Decimal Input with Range Display template is located in the Value Display & Input subfolder in the Control Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template when defining custom settings for displaying non-BOM content, and it is the default for the Decimal Features setting.

Item Selection Tables with Quantity and Range Display

When the FCE is enabled, the following Item Selection Table templates are available:

- Single Select BOM Item Table with Quantity and Range Display
- Single Select BOM Item Table with Header, Quantity and Range Display
- Multi-Select BOM Item Table with Quantity and Range Display
- Multi-Select BOM Item Table with Header, Quantity and Range Display
- Single Select Counted Options Table with Quantity and Range Display
- Multi-Select Counted Options Table with Quantity and Range Display

The content and appearance of these templates is very similar. Each template consists of a table with columns labeled Select, Quantity, Item, and Description, and each option's valid input range (domain) is displayed in the Quantity column (for example, "Range: 0 to 10").

The single-select templates contain a radio button in the Select column, while the multi-select templates provide a check box.

Which template is used by default depends on whether the Model node allows only one or multiple of its child options to be selected, and whether the node is a or a BOM item (for example, a BOM Option Class).

The Content Templates listed above for single and multi-select Counted Options are *not* used by default in any UI Master Templates. The default templates do not include the item's valid input range at runtime, so if you want to display the input range for Counted Options, create a custom UI Master Template and specify one of the templates listed above for the Single Select with Option Quantity or Multi-Select with Option Quantity setting.

Displaying BOM Relative Quantity at Runtime

For a definition of the term "relative quantity", see BOM Nodes, page 3-22.

Procedure

The BOM Item Selection Table templates listed in this section display each BOM item's *absolute* Quantity, Minimum Quantity, and Maximum Quantity. If you want to display a BOM item's *relative* quantities at runtime:

1. Copy one of the predefined BOM Item Selection Table templates listed above, and then open it for editing.

2. Open the 'Quantity' Text Input element for editing, and then bind it to `AssociatedModelNode.RelativeQuantity()`.
3. Set the Text Expression for the range display hint to "Range: &MinRelQuantity to &MaxRelQuantity".
4. Click **Display Condition**, and then select **Associated Model Node** as the Object, and select `IsBoundRelQuantity` from the **Property** list.
5. In the Condition section, select **Is** and set **Value** to False.

The full definition of the display condition is:

```
AssociatedModelNode.IsBoundRelQuantity() Is False
```

6. Save the changes.

BOM Item Status Region with Quantity and Range Display

This template displays a BOM item's current status and valid input range (domain). For example:

```
Status: Selectable    Quantity _____ (Range: 0 to 10)
```

Utility Templates

The UI Content Templates in this section are available when you are working in a custom UI Master Template and are selecting templates in the Utility Templates section.

Auto-Complete Status Dialog

For an overview of Auto-Complete, see Auto-Complete, page 1-3.

The Auto-Complete Status Dialog appears at runtime after Auto-Complete has finished processing. This page appears when Auto-Complete is invoked:

- By explicit end-user action, such as by clicking the Auto-Complete button
- By implicit end-user action, such as by clicking the Finish action, when the configuration does not contain any items that require end-user input.

In this case, the Auto-Complete Status Dialog appears only when Auto-Complete:

- Fails
- Results in changes to the orderable configuration
- Results in the addition of new Text Features that require end-user input (because such Features cannot be resolved by Auto-Complete)

or

- Results in new validation failures (such as from a post-Auto-Complete Configurator Extension)

If Auto-Complete is successful, then the Auto-Complete Status Dialog provides a summary of the configuration and indicates any changes that were made (an icon appears next to each changed item in the Configuration Summary table).

If Auto-Complete succeeds but the process created one or more required Text Features or invalid items, then the page lists all items that require end-user input or have validation failures. Each incomplete or invalid item appears as a link that the end user can use to navigate to the page containing the item.

If Auto-Complete fails, then this page displays a message similar to the following:

Error

The configuration could not be completed. A solution could not be found that is compatible with all of your selections. Please make some changes and try again.

To learn which options are available to the end user in each scenario, see Auto-Complete Status Button Bar, page 3-57.

The Auto-Complete Status Dialog template is located in the Utility Templates folder in the Main Area of the Repository. This template is the default for the Auto-Complete Status setting in a predefined UI Master Template. In a custom UI Master Template, you can select this template for use in the Utility Templates section.

Instance Chooser Page

This template allows the end user to select an existing, unassigned instance to add to an Instance Set. The Instance Chooser Page is displayed when the user invokes the Add Instance or Add and Go to Instance action and there are unassigned, configured, or connected instances of the same type in the instance pool (see The Unassigned Instance Pool, page 4-7). In this case, the end user selects an instance from the list, and Oracle Configurator adds it to the Instance Set.

If there are no instances of the same type in the instance pool when an end user invokes either Add Instance or Add and Go to Instance, then Oracle Configurator does not display the Instance Chooser Page. In this case, Oracle Configurator simply creates an entirely new instance (and navigates to the new instance, when Add and Go to Instance is invoked).

The predefined Instance Chooser Page template is located in the Utility Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Utility Templates section, and it is the default for the Instance Chooser setting.

For more information, see Instance Management, page 4-4.

Connection Management Table

This template displays a list of existing connections for a Multiple-Instance Connector (see Connectors, page 3-24). It includes controls that enable the end user to add new

connections and remove (disconnect) existing connections.

This template is the default for the Multiple-Instance Connectors setting in a UI Master Template.

Message Templates

The UI Content Templates in this section are located in the Message Templates folder in the Main area of the Repository. They are also available when you are working in a custom UI Master Template and are selecting templates in the Message Templates section.

Input Required Message Box

This template appears when an end user clicks Finish at runtime and at least one item requires end-user input. It displays following message at the top of a containing one or more required, invalid, or unbound options:

```
Input Required
Please complete all required inputs before finishing the configuration.
```

An icon also appears next to each item that requires end-user input.

This template is the default for the Input Required on Finish setting, which is available in all UI Master Templates and in the UI Definition.

For more information, see:

- Require End-User Input Setting, page 3-19
- Finish Configuration, page 4-9

The predefined Input Required template is located in the Message Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Message Templates section, and it is the default for the Input Required on Finish setting.

Input Required Dialog Page

This template is the default for the Input Required on Auto-Complete setting in the UI Master Template and in the UI Definition.

This template displays a page containing links to each option that requires input before the end user can invoke the Auto-Complete process. The end user can navigate to each option that requires input using links provided, click a button to return to the configuration, or save the configuration for later.

The text that appears on this page by default is similar to the following:

```
The configuration cannot be completed at this time. The following items
require your input:
```

```
Basic Options : How much memory do you require?Basic Options : Do you
want to be able to copy DVDs?Basic Options : Do you want a flat screen
monitor?
```

For more information, see [Require End-User Input Setting](#), page 3-19

The predefined Input Required Dialog Page template is located in the Message Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Message Templates section, and it is the default for the Input Required on Auto-Complete setting.

This template references the Input Required Dialog Button Bar, page 3-56.

User Request Conflict Dialog Page

This message is displayed when a conflict at runtime is the direct result of an end user's action, such as selecting an option or entering a value. (The term conflict is defined in [Conflict Handling and Resolution](#), page 4-9.) The Auto-Override for Conflicts setting in the UI Definition controls whether this message includes the consequences of an override. For example:

```
Confirming this action will undo the following:  
* Select RAM 1GB  
* Set 'Will you use this computer for home use?' to 'No'
```

When Auto-Override is enabled, clicking the OK button (Override action) returns the end user to the configuration page where the conflict occurred. When Auto-Override is disabled, the message contains a Continue button instead of an OK button, and clicking Continue displays the Confirm Override Dialog Page, page 3-55.

For more information about which buttons may be displayed at runtime when a User Request Conflict occurs, see [Basic Conflict Button Bar](#), page 3-58.

The predefined User Request Conflict Dialog Page template is located in the Message Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Message Templates section, and it is the default for the User Request Conflict setting.

For more information about Auto-Override, see [Auto-Override](#), page 4-10

Fundamental Conflict Message

For background information, see [Fundamental Conflict](#), page 4-11.

This message appears when launching a configuration when the root of the configuration model is overconstrained. Fundamental Conflicts should be discovered during unit testing and resolved before a configuration model is deployed in a production environment.

The text of the Fundamental Conflict Message is similar to the following:

```
Error in Configuration Model  
The model is overconstrained. It is not possible to continue.
```

The Fundamental Conflict Message template cannot be viewed or modified in Oracle Configurator Developer.

Confirm Override Dialog Page

This message contains information about how to resolve a conflict that was caused by

the end user's action (for example, selecting an option or entering a value). This message is displayed at runtime when Auto-Override is disabled and the end user clicks Continue in the User Request Conflict Dialog Page, page 3-55.

If the override succeeds, the Confirm Override message displays a list of the prior requests that were removed, and allows the user to confirm or cancel the request that caused the conflict. If the override fails, this message notifies the user that overriding the conflict is not possible. In this case, the end user has no choice but to cancel the override and then return to the configuration.

The predefined Confirm Override Dialog Page template is located in the Message Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Message Templates section, and it is the default for the Confirm Override setting.

For more information, see Auto-Override, page 4-10.

This template references the Confirm Override Button Bar, page 3-58.

Undo Status Message Box

This message appears at runtime when an end user clicks Undo after Auto-Complete has finished processing. The Undo Auto-Complete action rolls back all changes to the configuration that were made by Auto-Complete and returns the end user to the page from which Auto-Complete was invoked (either by clicking Finish or Auto-Complete).

The default text of this message is similar to the following:

```
Information:  
Undo was successful.
```

The Undo button appears in the Auto-Complete Status Dialog, page 3-52. See the description of that template for more information.

The predefined Undo Status Message Box template is located in the Message Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Message Templates section, and it is the default for the Undo Status setting.

Button Bar Templates

The UI Content Templates in this section are located in the Button Bar Templates folder in the Main area of the Repository. They are also available when you are working in a custom UI Master Template and are selecting templates in the Utility Templates section.

Input Required Dialog Button Bar

This template is used by the Input Required Dialog Page, page 3-54. This template consists of the following buttons:

- Return to Configuration: Returns the end user to the configuration session.
- Save for Later: Saves the incomplete configuration and ends the configuration

session.

The predefined Input Required Dialog Button Bar template is located in the Button Bar Templates folder in the Main Area of the Repository. This template is not available for use in a UI Master Template.

Auto-Complete Status Button Bar

This template is used by the Auto-Complete Status Dialog, page 3-52. It consists of buttons that appear after Auto-Complete has finished processing. The buttons that this template displays varies based on the result of the Auto-Complete process.

For example:

- If Auto-Complete succeeds and the configuration is complete, then the following buttons are displayed: Undo Auto-Complete; Return to Configuration; Finish.
Clicking Undo returns the configuration to the state it was in before running Auto-Complete. Clicking Return to Configuration enables the end user to modify the completed configuration. Clicking Finish saves the configuration and returns the end user to the host application.
- If Auto-Complete succeeds but the configuration is not complete, then the following buttons are displayed: Undo; Return to Configuration; Save for Later.
- If Auto-Complete fails, then the following buttons are displayed: Return to Configuration; Save for Later.

The predefined Auto-Complete Status Button Bar template is located in the Button Bar Templates folder in the Main Area of the Repository. This template is not available for use in a UI Master Template.

Tip: By default, the Auto-Complete Status Button Bar - and by extension the Undo Auto-Complete button - appears only in the Auto-Complete Status Dialog. If you want to add an Undo Auto-Complete button to a custom UI template, define a display or enabled condition for the button that uses the System Property `InAdjustMode`. For example, if you define the following display condition, then the button is displayed only when `ConfigurationSession.InAdjustMode` is `True`:

```
Object: Session Data
Property: InAdjustMode
Comparison: Is
Value: True
```

Processing Page Button Bar

This template is used only by the Generic Processing Page with Stop Button template, page 3-60. It consists of a Stop button that the end user can use to cancel a long-running

process. For details, see *Generic Processing Page with Stop Button*, page 3-60.

The predefined *Cancel Processing Button Bar* template is located in the *Button Bar Templates* folder in the *Main Area* of the *Repository*. Since the processing page templates cannot be customized, you cannot use this button bar in your own custom *User Interfaces*.

Basic Conflict Button Bar

This template is used by the *User Request Conflict Dialog Page* template, page 3-55. The buttons that are displayed at runtime may change depending on whether *Auto-Override* is enabled and, if it is, whether the override was successful.

For example:

- If *Auto-Override* is enabled and the override succeeds, then the following buttons are displayed: *OK*; *Cancel*.

Clicking *OK* overrides the conflict, while clicking *Cancel* cancels the request.

- If *Auto-Override* is enabled and the override fails or the conflict could not be overridden, then only a *Cancel* button is displayed.

- If *Auto-Override* is disabled, then the following buttons are displayed: *Continue*; *Cancel*.

Clicking *Continue* overrides the conflict, while clicking *Cancel* cancels the request.

The predefined *Basic Conflict Button Bar* template is located in the *Button Bar Templates* folder in the *Main Area* of the *Repository*.

Confirm Override Button Bar

This template is used by the *Confirm Override Dialog Page*, page 3-55. If the override was successful, an *OK* and a *Cancel* button are displayed. The end user can either click *OK* to confirm the message, or *Cancel* to cancel the override request.

If the override failed, only a *Cancel* button is displayed, and the end user's only option is to cancel the request.

The predefined *Confirm Override Button Bar* template is located in the *Button Bar Templates* folder in the *Main Area* of the *Repository*.

Instance Chooser Button Bar

This template is used by the *Instance Chooser Page*, page 3-53. It consists of *Cancel* and *Apply* buttons that enable the end user to either cancel or confirm the action of adding an instance to an *Instance Set*.

The predefined *Instance Chooser Button Bar* template is located in the *Button Bar Templates* folder in the *Main Area* of the *Repository*.

Preview Page Button Bar

This template displays the buttons that appear in the Configuration Summary page, and it is used by the Configuration Summary templates. The buttons that are displayed at runtime depend on the status of the configuration.

For example:

- If the configuration is complete, then the following buttons are displayed: Return to Configuration; Finish.

For details about the Finish button and related configuration flow, see Finish Configuration, page 4-9.

- If the configuration is incomplete, but there are no invalid items or items that require input, then the following buttons are displayed: Return to Configuration; Finish; Save for Later.

Clicking the Save for Later button saves the incomplete configuration and ends the configuration session.

- If the configuration is incomplete, but there are invalid items or items that require input, then the following buttons are displayed: Return to Configuration; Save for Later.

The predefined Preview Page Button Bar template is located in the Button Bar Templates folder in the Main Area of the Repository. In a custom UI Master Template, you can select this template for use in the Utility Templates section, and it is the default for the Preview Page Button Bar setting.

Other Templates

The UI Content Templates in this section are available when you are working in a custom UI Master Template and are selecting templates in the Other Templates section.

Generic Processing Page

For background on the use of processing pages, see Displaying a Processing Page at Runtime, page 3-64.

This is the default template for the UI actions that support processing pages. Those UI actions are listed in Displaying a Processing Page at Runtime, page 3-64.

The page template includes a default "busy" icon and the following text elements, in which you enter text that gives the end user information about the long-running action:

- Page Title: This text appears in the title bar of the window displaying the processing page, and in the page heading.
- Main Message: This text appears above the busy icon. The text is rendered in boldface.

- **Processing Caption:** This text appears below the busy icon.

There is no default text for these text elements. You must provide text, or the text elements will be empty at runtime.

Apart from the custom text described here, you cannot customize the appearance of the processing page.

The predefined Generic Processing Page template is located in the Other Templates folder in the Main Area of the Repository.

Generic Processing Page with Stop Button

For background on the use of processing pages, see *Displaying a Processing Page at Runtime*, page 3-64.

This template is the same as the Generic Processing Page, page 3-59, but it includes a Stop button that enables the end user to interrupt the Auto-Complete Configuration action and return to the runtime UI from which it was invoked, without changing the configuration. The Stop button undoes all of the selections made by the current Auto-Complete action.

This template can be specified by selecting it as the Processing Page Template for one of the UI actions that support processing pages. Those UI actions are listed in *Displaying a Processing Page at Runtime*, page 3-64. (The default template for such UI actions is the Generic Processing Page.)

This template is intended for use with larger or more complex FCE Models that might require longer processing.

The predefined Generic Processing Page with Stop Button template is located in the Other Templates folder in the Main Area of the Repository.

Changes to Existing User Interface Content Templates

This section describes how existing UI Content Templates change after converting a Model to use the FCE.

Configuration Summary Templates

When you convert a Model to use the FCE, the Combination Status region in all Summary Page UI Content Templates is replaced by a Messages region. This region appears at the top of the page and it contains textual information regarding the status of the configuration. For example, "No required or invalid items," "The configuration is not complete," or "The following items require your input before finishing the configuration." In the latter example, each item appears as a link that the end user can click to go directly to the page containing that item.

The lower portion of the Summary Page displays BOM-related nodes selected during the session. Any item that is in the Configuration Summary which is selected but has an unbound Quantity displays the range of the Quantity instead of a value.

When a Model uses the FCE, any selected options that represent non-BOM nodes do not appear in the Configuration Summary page. This is true whether or not the Orderable setting is selected in a node's details page in Configurator Developer. For details about this setting, refer to the section on orderable items in the *Oracle Configurator Developer User's Guide*.

Layout Regions

Instance List Layout Regions for OCE Models are described in the *Oracle Configurator Developer User's Guide*. An Instance List Layout Region displays content from optional or instantiable component instances on a Page that is the parent of those instances. There are some differences in the way that you use an Instance List Layout Region with FCE Models.

- At runtime, an Instance List for an FCE Model can include placeholders, which also occur in Instance Management Tables for FCE Models. (For information on placeholders, see UI Actions in Instance Management Tables, page 4-5.) Since placeholders usually represent generic instances (which do not have any children), there are no child node details available to display on the parent page of the Instance List. Trying to show the child node details for a placeholder instance in an Instance List region can result in unexpected behavior at runtime. Consequently, you should define, within your Instance List region, a Switcher Region that contains two Case Regions, each based on a Switcher condition using the Associated Model Node's property `IsPlaceholder`. Then define these Case Regions to display an appropriate set of UI elements depending on whether or not the instance is a placeholder. If `IsPlaceholder` is true, that Case Region should only include the appropriate controls for a generic instance, such as Configure and Delete. If `IsPlaceholder` is false, then that Case Region can include other details for the instance.
- The Configure (Drilldown to SubComponent) action behaves differently for a placeholder instance under an Instance List Layout Region. When the user chooses Configure, it activates the placeholder. If there are instances available in the instance pool, then the Instance Chooser page is presented. When an instance is chosen from the Instance Chooser, the user returns to the Instance List on the parent page, rather than to the page for the instance itself (as happens in an Instance Management Table). If there are no available instances, then the placeholder is replaced with a new identifiable instance, and remains in the Instance List on the parent page.

User Interface Elements

This section describes UI elements that are available only when using the FCE.

Instance Chooser Table

This element displays all instances (configured or connected) that can be added to an

Instance Set. You can create this element in a UI Page, an Instance Management Template, and a Generic Template. This table is also included by reference in the Instance Chooser Page, page 3-53.

This element's Associated Model Node must be an instance set.

Multiple-Instance Connector Control

This element represents a Multiple-Instance Connector at runtime (for details, see Connectors, page 3-24).

You can create this element in a UI Page, a Connector Control Template, or a Generic Template. This element's Associated Model Node must be a Connector.

User Interface Actions

The table below describes the UI actions that are available when using the FCE.

User Interface Actions

Action	Description
Add Instance	See UI Actions in Instance Management Tables, page 4-5 for a description of this action.
Auto-Complete Configuration	<p>Invokes Auto-Complete (the FCE search procedure) to bind all variables in the configuration and present a solution to the end user.</p> <p>This action is valid only when the end user is not in a nested transaction or a pending mode (for example, conflict processing).</p> <p>For details, see Auto-Complete Configuration, page 4-2.</p>
Cancel Processing	<p>Cancels a long-running process (for example, Auto-Complete Configuration).</p> <p>Cancels a long-running process (for example, Auto-Complete Configuration). This action is valid only in a predefined template that supports the processing page (for example, the Generic Processing Page Template). You cannot use this action in your own custom User Interfaces.</p> <p>For more information, see Auto-Complete Configuration, page 4-2.</p>

Action	Description
Cancel Request	<p>Rolls back any conflict override, cancels the current/original user request, and returns to the configuration.</p> <p>This action is valid while processing a conflict.</p> <p>For more information, see Conflict Handling and Resolution, page 4-9.</p>
Copy Instance	See UI Actions in Instance Management Tables, page 4-5 for a description of this action.
Configure Instance	See UI Actions in Instance Management Tables, page 4-5 for a description of this action.
Create Instance	<p>Creates a new instance and adds it to the current instance set, as defined by the Associated Model Node of the UI element assigned to the action. After invoking this action, Oracle Configurator does <i>not</i> present the end user with the option to add an existing instance from the instance pool.</p> <p>For details, see Create Instance UI Action, page 4-8.</p>
Create and Go to Instance	<p>Creates a new instance and adds it to the current instance set, then navigates to the UI page that represents that instance.</p> <p>After invoking this action, Oracle Configurator does <i>not</i> present the end user with the option to add an existing instance from the instance pool.</p> <p>For details, see Create and Go to Instance UI Action, page 4-8.</p>
Delete Instance	See UI Actions in Instance Management Tables, page 4-5 for a description of this action.
Override Conflict	<p>If Auto-Override is enabled and override was successful, this action commits the override and returns the end user to the configuration. If Auto-Override is disabled, it invokes the override and displays an override status message.</p> <p>For more information, see Auto-Override, page 4-10.</p>
Remove Instance	See UI Actions in Instance Management Tables, page 4-5 for a description of this action.

Action	Description
Save and Exit	<p>Saves and exits the configuration without invoking the Auto-Complete process. This action can be used whether or not the configuration is complete.</p> <p>This action is valid only when the end user is not in a nested transaction and the configuration is not a "pending" mode (for example, processing a conflict).</p>
Undo Auto-Complete	<p>Rolls back all changes made to the configuration as a result of running Auto-Complete, and returns the end user to the page from which Auto-Complete was invoked.</p> <p>In order to perform the Undo Auto-Complete action, the configuration session must return True for one of the system properties <code>AutoCompleteSuccessful</code> or <code>InAdjustMode</code>.</p> <p>For details, see Auto-Complete Configuration, page 4-2.</p>

Displaying a Processing Page at Runtime

Some UI actions take longer than others to complete at runtime. For example, the Auto-Complete Configuration action may take several seconds or more before returning control to the end user. When such longer-running actions are invoked, you may want to display a "processing" page to inform your end users that a process is running in the background, and that they will be able to proceed when it is complete.

In Configurator Developer, you can choose which processing page to display at runtime when you assign certain UI actions to a UI element. The UI action must support the use of a processing page. The UI actions that support processing pages are the following:

- Auto-Complete Configuration

This action is only available for FCE Models.

- • Apply/Finish/Confirm

The Finish action supports processing pages.

Note: The processing page is only available for FCE Models. No processing pages are supported for Original Configurator Engine (OCE) Models.

The processing page is displayed at runtime when the following are true:

- The end user invokes a supporting UI action in which the Display Processing Page

setting is enabled

- The amount of processing time for the action exceeds the limit previously specified by the profile option CZ: Processing Page Delay, page 2-3.

There is no default processing page for the Auto-Complete Configuration action. A default processing page (with no Main Message) is displayed for a long-running Finish action when you click an uncustomized Finish button.

If the processing for an action does not run long enough to invoke the processing page, then the page ordinarily displayed for the action appears. The Auto-Complete Configuration action ordinarily produces the Auto-Complete summary page, which allows the end user to undo the Auto-Completion, or to Return To Configuration in Adjust Mode. The Finish action ordinarily produces a configuration Confirmation page. However, Finish can invoke Auto-Complete on an incomplete configuration that does not contain any items that are invalid or require end-user input; in that case Finish displays the Auto-Complete summary page.

Procedure

To specify a processing page for a UI action:

1. Edit the details page for a UI element (such as a Custom Button).
2. For the element's Action, click **Define**.
3. In the UI action's Choose Action (or Define Action) page, under Session Control, choose one of the actions that supports the processing page. The supporting actions are listed elsewhere in this section.
4. Expand the **Processing Page** section.

The Processing Page section only appears when you select actions that support the processing page.
5. Select **Display Processing Page**, to enable the use of the Processing Page.
6. Optionally, for **Processing Page Template**, click **Choose** to select a different template.
 - You can accept the default template, which is the Generic Processing Page, page 3-59.
 - You can select the predefined Generic Processing Page with Stop Button, page 3-60 template.
 - You can select any customized versions of those Processing Page templates that you have created.
7. Optionally, enter custom text for **Page Title**, **Main Message**, or **Processing Caption**.

See Generic Processing Page, page 3-59 for an explanation of these text elements on the template page.

Tip: The Main Message and Processing Caption are Formatted Text elements, so you can use simple HTML markup to format the text. See the *Oracle Configurator Developer User's Guide* for a list of HTML tags supported in Formatted Text elements.

8. Click **Apply**.






Your settings related to the processing page are retained when you copy or move a UI element



Runtime Icons and Images

The table below lists the icons and images that are used to indicate an option's selection state and specific UI actions at runtime.

For details about how to change the default images that indicate selection state at runtime, see User Interface Master Templates, page 3-48.

Icons and Images Used at Runtime

Name	Description	Icon
Recommended Check Box (recommended_checkbox.gif)	An enhanced Check Box image for the 'Proposed Selected' state.	
Recommended Radio Button (recommended_radio_button.gif)	An enhanced Radio Button image for the 'Proposed Selected' state.	
Recommended Status (recommended_status.gif)	A status icon for the 'Proposed Selected' state.	
Not Recommended Check Box (not_recommended_checkbox.gif)	An enhanced Check Box image for the 'Proposed Excluded' state.	
Not Recommended Radio Button (not_recommended_radiobutton.gif)	An enhanced Radio Button image for the 'Proposed Excluded' state.	

Name	Description	Icon
Not Recommended Status (not_recommended_status.gif)	A status icon for the 'Proposed Excluded' state.	
Proposed Status (proposed_status.gif)	A status icon that indicates the associated item's state or value is the result of either a Default or Search Decision.	

User Interface Definition

A User Interface's UI Definition contains information and settings that are specific to the FCE when the Model-level Configurator Engine setting is set to *Fusion* (in other words, when the UI belongs to an FCE Model).

In this case, the UI Definition also contains the Enable Auto-Override for Conflicts setting. The default value of this setting is determined by the Master Template that was used to generate the UI, but you can change it in the UI Definition. For more information, see *Auto-Override*, page 4-10.

Note: Auto-Override is always enabled in Models that use the Original Configurator Engine.

The Auto-Override for Conflicts setting also determines the value of the session property `AutoOverrideEnabled` at runtime. For details about this property, see *Configuration Session Properties: Conflict Processing*, page 3-15.

When the UI belongs to an FCE Model, its UI Definition also contains settings that control which Button Bar, Utility Page, and Message UI Content Templates the UI uses at runtime. All templates and images that are used by default are determined by the UI Master Template that was used to generate the UI. You can select different templates and specify different images that are used to indicate options that have changed or still require end-user input.

Unit Testing a Configuration Model Using the Model Debugger

For background about unit testing and the Model Debugger, see the *Oracle Configurator Developer User's Guide*.

Most of the information in the *Oracle Configurator Developer User's Guide* also applies when testing an FCE Model, but there are some exceptions. This section describes additional functionality and features of the Model Debugger that are available only when testing an FCE Model.

The Edit Totals and Resources setting is disabled when you launch the Model Debugger to unit test an FCE Model.

Tip: Open an FCE Model for editing and then test the Model using the Model Debugger while reading this section. Then, navigate to each area of the Model Debugger and note the icons, UI labels, and functionality described in the following sections.

Configuration Tab

When testing an FCE Model, the Model Debugger displays unique icons that indicate logic states that are available only in the FCE and other icons that identify whether an option was selected as a result of a Search Decision or by a Default.

Columns

The Configuration tab may contain the following additional or enhanced columns for FCE Models:

- **Logic State:** This existing column displays new icons which indicate logic states that are available only in the FCE, and other new icons that identify whether an option was selected as a result of a Search Decision or by a Default. See Runtime Icons and Images, page 3-66
- **Input Required:** This column replaces the Unsatisfied column used for Original Configurator Engine Models. The presence of an icon in this column indicates whether a node requires a value. See Require End-User Input Setting, page 3-19.
- **Range:** When the associated node is unbound, this new column displays the node's minimum and maximum:
 - Value, for Numeric Features, Totals, and Resources
 - Quantity, for Feature Options and BOM nodes (all types)
 - Instances, for Components
 - Connections, for Connectors
 - Selections, for Option Features
 - Selections, for Boolean Features (False.. True)

When the associated node is bound, the Range column is blank.

To display this column in the Configuration tab, create a View and add it to the 'Columns Displayed' list.

This column appears in the Watch List by default.

- **Changed by Auto-Complete:** An icon appears in this new column if Auto-Complete modified the item's value or selection state.

Controls

The Configuration tab includes the following UI controls for FCE Models:

- **Auto-Complete:** This button invokes Auto-Complete, page 4-2. After Auto-Complete finishes processing, all Model interaction UI controls are read-only. To modify the configuration, click Adjust Configuration. To return the configuration to its previous state, click Undo Auto-Complete.

If any items in the configuration require end-user input, the Model Debugger displays a message and lists the invalid or required items. After addressing all invalid or required options in the configuration, you can invoke Auto-Complete again.

The Auto-Complete Configuration button appears in the Configuration tab by default.

- **Adjust Configuration:** After running Auto-Complete, the configuration is in read-only mode and you must click this button to make any changes to the configuration.

You can also click Undo Auto-Complete while modifying a completed configuration, or click Auto-Complete Configuration again after making changes.

- **Undo Auto-Complete:** This button is available after you click Auto-Complete Configuration, and it enables you to return the configuration to the state it was in before running Auto-Complete.

If a Connector allows multiple connections, the Value column in the Configuration tab contains the message "See Node Details." Click the Connector's name to view its details page and all of the connected target instances. For more information see Model Node Details Pages, page 3-69.

If any instances of the same type exist in the instance pool, clicking the icon in the Add Instance column in the same row as an instantiable component displays the Instance Chooser Page. In this case, you can either create a new instance or select one from the list unassigned instances. If no instances of the same type exist in the instance pool, clicking the icon in the Add Instance column creates a new instance of the component.

If the configuration is incomplete and does not contain any items that are invalid or require end-user input, clicking the Finish button invokes Auto-Complete. For more information, see Finish Configuration, page 4-9.

Model Node Details Page

To display a Model node's details page, click the node's name in the Configuration tab.

The details page for each node includes information about the node's domain, such as

the minimum and maximum value, quantity, selections, instances, and so on. Any Configurator Extension command events that are associated with the node also appear on a node's details page.

A Connector node's details page lists all component instances that are currently connected and all target instances that are available for connection.

A Connector node's details page also indicates:

- Whether the Connector is a Reverse Connector
- How many connections currently exist
- The Minimum and Maximum Connections specified for the Connector in Configurator Developer (for Multiple-Instance Connectors)

The details page of any component that is the target of a Connector lists all currently connected components (in the 'Connections to this Instance' table).

Summary Tab

In the Summary tab, an icon appears in the Status column to indicate how each node was added to the configuration. Unique icons indicate whether each item was added by the end user or by the propagation of a constraint, or if the item is a proposed selection (that is, it was added by either a Search Decision or a Default). To view the description of each icon, click Show Legend.

The Summary tab shows only items within the configuration that are orderable (that is, included on a sales order); therefore, any non-BOM items are not included.

If Auto-Complete was invoked during the unit testing session, then the Summary tab includes the Adjust Configuration and Undo Auto-Complete buttons. Click Adjust Configuration to modify the completed configuration. For more information about Undo Auto-Complete, see Undo Auto-Complete, page 3-64.

Status Tab

In addition to the sub-tabs described in the *Oracle Configurator Developer User's Guide*, the Status tab contains the following sub-tabs: Input Required, Proposed Items, and User Requests.

The Input Required sub-tab displays all Features, Connectors, BOM Option Classes, and Text Features in the configuration that require end-user input. This column also appears in the Watch List table by default.

The Proposed Items sub-tab displays all items that Auto-Complete added to the configuration and items whose values were provided by Auto-Complete.

The User Requests sub-tab lists each item that you added to the configuration; items added to the configuration by Auto-Complete do not appear. This area also describes the action that occurred to add the item to the configuration. For example:

Name	Request
Integer Feature X	Set "Integer Feature X" to '27'
BF1	Select 'BF1'

Note: If you upgraded from a previous release of Oracle Configurator, you may notice that the Status tab no longer contains an Unsatisfied Rules tab. This is because rules cannot be unsatisfied in an FCE Model.

Runtime Behavior of the Fusion Configurator Engine

This chapter covers the following topics:

- Domain Display and Availability
- Logic State Display
- Runtime Configurator Flows and Behavior

Domain Display and Availability

For background information, see Domain Ordering Setting, page 3-16.

As an Oracle Configurator end user makes selections that affect a node's specified domain, Oracle Configurator dynamically updates the domain range that is displayed in the UI. For example, if an Integer Feature's domain is set at 1-10 in Configurator Developer, then its range appears as 1-10 at runtime. If the end user or the propagation of a constraint makes a selection that cause the values 6-10 to be invalid, then the Feature's range changes to "1-5".

When a variable (node) is bound at runtime, its range no longer appears. This is true regardless of whether it was bound by the end user or by the propagation of a constraint.

Logic State Display

At runtime, distinctive icons appear in the Status column of the Configuration Summary page to identify end user selections, system selections (options selected by rule propagation), and Proposed selections (options selected either by a Default or by Search Decisions, or by Auto-Complete). Proposed selections have a logic state of Recommended (Proposed Selected).

Options may also be *excluded* from a configuration by Defaults or Search Decisions, or

by Auto-Complete. These options have a logic state of Not Recommended (Proposed Excluded), and are also indicated by a distinctive icon during a runtime configuration session. For details, see *Runtime Icons and Images*, page 3-66.

See also *Logic States*, page 3-27 and *Selection State*, page 3-16.

Runtime Configurator Flows and Behavior

This section describes end user actions and runtime behavior that is specific to FCE Models. It includes the following sections:

- Auto-Complete Configuration, page 4-2
- Instance Management, page 4-4
- Finish Configuration, page 4-9
- Conflict Handling and Resolution, page 4-9
- Restoring a Completed Configuration, page 4-11

Auto-Complete Configuration

By default, UIs that you generate in Configurator Developer provide a button labeled **Finish**. Clicking this button in an FCE Model invokes Auto-Complete when the configuration:

- Is incomplete
- Does not contain any items that are invalid or require end-user input

When none of the options in the configuration require end-user input and the end user invokes Auto-Complete, the process applies all search decisions that you have defined in Configurator Developer, binds all variables in the configuration (except Text Features), and presents a completed configuration to the end user. For details about the page that appears when Auto-Complete finishes processing, see *Auto-Complete Status Dialog*, page 3-52.

After reviewing a configuration that was created by Auto-Complete, an end user can do one of the following:

- Save the configuration and return to the host application (by clicking Finish)
- Undo the results and continue making selections manually (by clicking Undo Auto-Complete)

See the Undo Auto-Complete action in *User Interface Actions*, page 3-62.

- Modify the completed configuration (by clicking Return to Configuration, which

places the user into Adjust Mode)

In order to perform the Undo Auto-Complete action, the configuration session must return True for one of the system properties `AutoCompleteSuccessful` or `InAdjustMode`.

It is also possible that Auto-Complete will be unable to find a solution. For details, see *Aspects of Auto-Complete Behavior*, page 4-3.

If an end user clicks Finish when one or more options in the configuration require end-user input, then Auto-Complete does not run. This can occur, for example, when the configuration contains a required Text Feature, which can only be bound by the end user. In this case, Oracle Configurator displays a message at the top of a page that contains at least one invalid or required option. After addressing all invalid or required options in the configuration, the end user can invoke Auto-Complete again, by clicking Finish.

When a configuration contains options that require end-user input, Oracle Configurator displays the Input Required Message Box, page 3-54 by default. You can display different content by specifying a different UI Content Template for the Input Required on Finish setting in your UI Master Template (before creating the UI), or in the UI Definition (after creating the UI).

Important: To invoke Auto-Complete from a custom UI page, you must add a control that invokes the Auto-Complete UI action. For example, create a Custom Button and specify Auto-Complete Configuration as its associated UI action.

Aspects of Auto-Complete Behavior

There are aspects of the runtime behavior of Auto-Complete that may appear to be limitations to its ability to complete a configuration.

- There may be some options in your Model that you have specified must be bound by the end user (see *Require End-User Input Setting*, page 3-19). If the end user clicks Finish (or invokes Auto-Complete using a custom UI control) when one or more options require end-user input but cannot be bound by Auto-Complete, Oracle Configurator displays a message listing those option(s). This behavior is by design, and is intrinsic to the capabilities of the FCE.
- A blank Text Feature may exist in a completed configuration only when it does not require a value. Consider this behavior carefully if you use custom UI templates, and test the UI thoroughly to ensure that end users can manually create a complete configuration by entering values for all required blank Text Features. For more information, see *Text Features*, page 3-19.
- It is possible that Auto-Complete will be unable to find a solution to the configuration problem. This can occur, for example, when a combination of Model

Constraints and end-user selections prevents Auto-Complete from binding one or more variables. In this situation, Oracle Configurator displays a message suggesting that the end user modify some previous selections and then re-invoke Auto-Complete. The scenarios in which Auto-Complete may be unable to create a complete and valid configuration are described in Conflict Handling and Resolution, page 4-9.

- For important information about Configurator Extensions and Auto-Complete, see Configurator Extensions, page 3-34.

Instance Management

Important: See *Oracle Configurator Release Notes, Release 12.1.1* (On MetaLink, Oracle's technical support Web site) for background on important enhancements and changes to runtime instance management for the FCE.

This section describes actions and behavior related to managing component instances at runtime.

When an end user is configuring an FCE Model, Oracle Configurator automatically may create one or more instances if additional instances are required to satisfy a constraint. For example, your company sells network servers and equipment racks that hold up to 5 servers. To ensure that enough instances of the Rack component exist in the configuration, you define the following Statement Rule:

```
rack.instanceCount() = ceiling(server.instanceCount() / 5)
```

At runtime, when the end user specifies 2 Servers, Oracle Configurator creates 1 instance of the Rack component. If the end user specifies 7 servers, then Oracle Configurator creates 2 Racks, and so on.

Other examples of when an instance may be created automatically include:

- The minimum number of instances required is greater than the number of instances that currently exist in an Instance Set.
- The Model contains a required Connector, but no potential target instance for that Connector exists in the configuration. (This is projected functionality.)

For more information, see Add Instance UI Action, page 4-8.

Terminology for Instance Management

The following terminology is related to instance management.

- *generic instance:* A generic instance has no characteristics that identify it as being different from other instances of the same component. An instance that is added by

Oracle Configurator and not yet configured or renamed is considered generic. The FCE's dynamic instantiation capability can generate generic instances in response to constraints that require their creation. Constraints can also prevent the user from deleting such required generic instances.

- *identifiable instance*: An identifiable instance has an identity produced by some explicit end user action. An instance that is added, configured, or renamed by a user is considered to have an identity distinct from that of any other instances of the same component.
- *placeholder*: A placeholder is a UI element indicating that an instance is required in that location, but that no identifiable instance has been specified for it. A placeholder usually represents a generic instance. (Under some circumstances, a placeholder represents no instance at all, but this distinction is not communicated to the end user.)

UI Actions in Instance Management Tables

For working with FCE Models, the UI templates named Instance Management Table and BOM Instance Management Table provide the actions described in the following table.

UI Actions in Instance Management Tables

Action	Description
Placeholder rows	Placeholder rows represent required instances in the table. When a constraint increases (or decreases) the minimum number of instances of a component required in an instance set (its <i>minimum cardinality</i>), any additional instances that must be created to meet that minimum are represented by placeholder rows.
Note: This functionality is not a UI action, but is an essential complement to UI actions.	Generic instances added to the Instance Management Table for a component are represented by placeholder rows, which are visually distinguished by the automatically generated label "[To be Configured]".
	When the user selects a placeholder and chooses the Configure action, the placeholder is replaced by an identifiable instance. Note: The Auto-Complete action changes placeholders into identifiable instances.

Action	Description
Add Instance	<p>Adds an instance. The Add Instance action always results in an increase to the minimum cardinality, and does not replace any existing placeholder rows.</p> <p>If there are any existing candidate instances in the instance pool, the Instance Chooser is presented, allowing the user to select one of them to use, rather than creating a new instance. The Instance Chooser also provides a row labeled "[New Instance]"; selecting this row creates a new identifiable instance. If there are no candidates, then a new identifiable instance is created. If adding an instance causes a non-overridable conflict, then no instance is added, either generic or identifiable.</p>
Copy Instance	<p>Creates a copy of the selected instance and places it in the instance pool (thus making it available to the Instance Chooser). The instance copied into the instance pool is given a name of the form "Copy of <i>component_name</i>". If additional copies are made of the original instance, they are named with the form "Copy <i>n</i> of <i>component_name</i>", with incrementing values of <i>n</i>, to show the chronological order of the copies. The Copy Instance action is provided by an icon in the Copy column in the Instance Management Table UI templates that support Copy And Remove. The Copy icon is never displayed in placeholder rows, since no identifiable instance is there to be copied.</p>
Configure Instance	<p>Configures an instance. When this action is selected on a placeholder row, the placeholder is replaced with a new identifiable instance. This action maintains the cardinality of the instance set, by increasing the number of identifiable instances and equally decreasing the number of placeholders.</p>
Delete Instance	<p>Deletes an instance permanently, and also always reduces the cardinality of the set, leaving one or more fewer rows in the table, depending on the circumstances. Deleting a placeholder row only reduces the cardinality of the set, since there is no identifiable instance to delete.</p>
Remove Instance	<p>Removes an identifiable instance and replaces it with a placeholder row, but does not reduce the cardinality of the set. The removed instance is placed into the instance pool (thus making it available to the Instance Chooser). The Remove Instance action is provided by an icon in the Remove column in the Instance Management Table UI templates that support Copy And Remove. The Remove icon is never displayed in placeholder rows, since no identifiable instance is there to be removed.</p>

The Unassigned Instance Pool

At runtime, it is possible for instances to be removed from an Instance Set and automatically put into a collection of unassigned instances. (In this context, "unassigned" means the instance does not belong to an Instance Set and it is not attached to a Connector.) This collection of unassigned instances is not visible to the end user and is known as the "instance pool."

An example of how an instance can become unassigned is when one configuration rule causes an instance to be created (or "contained") in an Instance Set, and the resolution of a conflict with another configuration rule overrides the containment, thus putting the instance into the instance pool, unassigned to any Instance Set.

Each instance in the instance pool can be added to another Instance Set later in the configuration session. An instance cannot be reassigned to the Instance Set from which it was removed (except in very rare circumstances). See Instance Chooser Page, page 3-53.

Unassigned instances may be also created in a Connector and then later added to an Instance Set. Additionally, only configured instances are displayed, which means instances that contain user inputs (including non-defaulted instance names), or instances that are connected to a component within the configuration. (Configured instances are also called identifiable instances.)

Default Instance Names

When configuring a Model that uses the Original Configurator Engine (Release 12.0 and earlier), a newly created instance is given an instance number based on its order of creation in a particular Instance Set. A default instance name is generated from this number along with the Display Name of the Component or Model Reference that represents the containing Instance Set. An instance cannot move from one Instance Set to another when using the Original Configurator Engine; therefore the default instance name and number are always similar to the containing Instance Set. However, this is not necessarily true when using the FCE.

When using the FCE, the name of all referenced Model instances defaults to the Display Name of the root node of the parent Model, appended with a 1-based ordinal number. For example, Hub Model-1, Hub Model-2, and so on. The name of each instance is unique throughout the configuration, rather than within the specific container, and it does not change as the instance's status changes within the configuration (for example, when it is assigned to another Instance Set or is attached to a Connector).

Oracle Configurator uses a similar convention when naming Component instances, but in this case the default name of each instance is derived from the Component name (this is the same behavior as the Original Configurator Engine). Similar to referenced Model instances, the ordinal number assigned to each instance is unique among all instances of the Component in the configuration.

Add Instance UI Action

This action enables the end user to add an instance of a component to an Instance Set. An end user can create a new instance only if the number of instances in the configuration has not yet reached the Maximum Instances specified in Oracle Configurator Developer, or the propagation of a rule has increased the Maximum Instance value specified in Configurator Developer.

If any unassigned instances of the same type exist in the instance pool when the end user invokes this action, Oracle Configurator displays the Instance Chooser Page, page 3-53. The end user can then select an existing instance from a list and add it to the Instance Set. (The term "instance pool" is defined in The Unassigned Instance Pool, page 4-7.)

If no instances of the same type exist in the instance pool when an end user invokes the Add Instance action, then the Instance Chooser Page does not appear and Oracle Configurator creates an entirely new instance.

By default, the Add Instance UI action is assigned to a button that appears in the Instance Management Table UI Content Template.

Add and Go To Instance UI Action

This action is similar to the Add Instance UI action, but Oracle Configurator navigates to the new instance immediately after the end user adds it to the Instance Set.

For more information, see Add Instance UI Action, page 4-8.

Create Instance UI Action

If the maximum number of instances allowed in the configuration has not been reached, then invoking this action creates a new instance of the specified component.

This action does *not* allow the end user to select an existing instance of the same type from the instance pool and add it to an Instance Set. See Add Instance UI Action, page 4-8.

Create and Go to Instance UI Action

This action is similar to the Create Instance UI action, but Oracle Configurator navigates to the new instance immediately after the end user creates it. See Create Instance UI Action, page 4-8.

BOM Instantiation and Selection

When an end user is configuring an FCE Model and creates an instance of a BOM Model, the new instance is selected (added to the configuration) automatically. In Models that use the Original Configurator Engine, the end user must manually select a newly created BOM Model instance to add it to the configuration.

Finish Configuration

If the configuration is incomplete and does not contain any items that are invalid or require end-user input, clicking the Finish button invokes Auto-Complete. If Auto-Complete finishes successfully, and the process does not change any of the orderable items, does create any new instances containing required Text Features, and causes no new validation failures to occur, then the completed configuration is displayed in the Configuration Summary page.

At this point, the end user can click:

- Undo: Rolls back the changes made by Auto-Complete, returns the end user to the configuration
- Return to Configuration: Keeps the changes made by Auto-Complete and returns the configuration session to edit mode so the end user can make changes
- Finish: Saves and returns configuration data to the host application

Note: If the order has not yet been booked in the host application, then it is still possible for an end user to modify a configuration that was created by Auto-Complete. See Restoring a Completed Configuration, page 4-11.

If the configuration is incomplete but there are one or more items that are invalid or require end-user input, then by default Oracle Configurator displays the Input Required Message Box, page 3-54 when the end user clicks the Finish button. This message lists one or more items that are invalid or require end-user input, and is displayed at the top of the first page in the UI (based on the UI's navigation sequence).

Tip: You can display a different message or UI Content Template in this situation by specifying a different UI Content Template for the Input Required on Finish setting. This setting is available in:

- All UI Master Templates
- The UI Definition

For example, you may want to display the Input Required Dialog Page, page 3-54 or a custom message instead of the Input Required Message Box.

Conflict Handling and Resolution

In FCE Models, several types of conflicts can occur at runtime. The following sections

describe each type:

- User Request Conflict, page 4-10
- Hidden Conflict, page 4-11
- Fundamental Conflict, page 4-11

User Request Conflict

A User Request Conflict occurs when an end user's action conflicts with the constraints defined in the Model. By default, the consequences of overriding this type of conflict appear in the User Request Conflict Dialog Page, page 3-55.

When a User Request Conflict occurs, the end user can do one of the following:

- **Override:** When possible, this action applies the end user's request and removes all conflicting requests. Some requests are not overridable.

For details, see Auto-Override, page 4-10.

- **Cancel:** This action withdraws the request that caused the conflict, and returns the end user to the configuration.

Note: A User Request Conflict is similar to an overridable contradiction, which is a type of conflict that can occur when using the Original Configurator Engine. For details, see the *Oracle Configurator Developer User's Guide*.

Auto-Override

When a User Request Conflict occurs at runtime, Oracle Configurator displays the User Request Conflict Dialog Page, page 3-55. If Auto-Override is enabled, then this message displays the consequences of overriding the conflict. In this case, the end user can either override the conflict by clicking OK and return to the page where the conflict occurred, or click Cancel to undo the action that caused the conflict.

If Auto-Override is disabled, the end user must either click Continue to view additional information about how to resolve the conflict, or click Cancel to undo the action that caused the conflict. If the end user clicks Continue, Oracle Configurator displays the Confirm Override Dialog Page, page 3-55.

The Enable Auto-Override for Conflicts setting is available in all UI Master Templates, and it is enabled by default.

Note: This setting is available only if the profile option CZ: Enable Configurator Engine is set to either *Both* or *Fusion*.

Hidden Conflict

A Hidden Conflict occurs when Oracle Configurator is processing an end user's request, but the conflict is not caused by the request itself. This type of conflict is caused by an internal mechanism of the FCE that controls the assignment of instances to an Instance Set or to a Connector, and it is triggered when an end user's request is removed. For example, when the user clears a numeric value, deselects a prior selection, or modifies a numeric value or selection (which causes the old value or selection to be removed as an interim step).

Because the difference between a Hidden Conflict and a User Request Conflict is not obvious to the end user, and since the options for resolving each type of conflict are the same, Oracle Configurator displays the User Request Conflict Dialog Page, page 3-55 when a Hidden Conflict occurs.

Fundamental Conflict

A Fundamental Conflict results when the root Model, one of its child Models, or a Component, is overconstrained. This means that no valid solution can be found using the constraints as they are currently defined in the Model. To resolve a Fundamental Conflict, you must modify the constraints for the component that caused the conflict in Configurator Developer, and then retest the Model.

Good model development practice requires that a Fundamental Conflict should be detected during system testing, and resolved before deployment of your application. This type of conflict, if it occurs at runtime, requires an end user to either exit the configuration, or continue without including the component that caused the conflict.

If the conflict exists in the root model or one of its required child Models, the Fundamental Conflict message appears when you attempt to unit test a configuration model from Configurator Developer. See Fundamental Conflict Message, page 3-55. In this case, Oracle Configurator does not initialize the unit testing session and you return to Configurator Developer.

If the problem exists in an optional component, it is possible launch a unit testing session successfully, but the User Request Conflict Dialog Page appears when the conflict occurs (for example, when you select or choose to configure the optional component). In this case, the end user will be able to continue, but the component that caused the conflict will not be included in the configuration. See User Request Conflict Dialog Page, page 3-55.

Fundamental Conflicts are easily detected when unit testing a Model from Configurator Developer. Therefore, it is unlikely that an overconstrained Model would ever be deployed in a production environment.

Restoring a Completed Configuration

When an end user restores a completed configuration, all of the selections and values from the original session are retained and the end user can navigate the configuration,

select or deselect options, and change values. This is true regardless of whether Auto-Complete was invoked during the configuration session. An end user can also restore configurations that were created before a Model was converted to an FCE Model.

In a restored configuration, any instances that existed in the instance pool from the original configuration session are no longer available. This is because Oracle Configurator purges all instances in the pool when the end user saves the original configuration. For more information about the instance pool, see *The Unassigned Instance Pool*, page 4-7.

A restored configuration opens in a default state that is the equivalent of Adjust Mode, so it does not include an Undo Auto-Complete button. This button appears by default only in the Auto-Complete Status Dialog, which is displayed after Auto-Complete completes successfully.

Procedure

If you want end users to be able to return a configuration to the state it was in before running Auto-Complete, then perform the following before publishing the Model:

1. Open the UI for editing in Configurator Developer.
2. Create a UI control, such as a button, on the UI's first page. (Which page appears first at runtime depends on the UI's primary navigation style.)
3. In the UI control's details page:
 - Assign the Undo Auto-Complete UI action.
 - Define a display condition that uses the `InAdjustMode` System Property.

For example:

```
Object: Session Data
Property: InAdjustMode
Comparison: Is Condition: True
```

4. Click **Apply**.

The UI control created in this example appears only when an end user is modifying a restored configuration.

Configuration Attributes for Fusion Configurator Engine Models

This chapter describes how to set up configuration attributes for Models that use the Fusion Configurator Engine.

This chapter covers the following topics:

- About Configuration Attributes
- Tasks for Adding Configuration Attributes to an FCE Model
- Setting Up Descriptive Flexfields
- Adding Attribute Features
- Associating Attribute Features to Flexfield Segments
- Associating BOM Nodes with Attribute Features
- Defining the Configurator Extension Rule
- Access to Configuration Attribute Data
- Special Considerations
- Maintaining the Configuration Attributes Setup
- Using Configuration Attributes in the Downstream Application

About Configuration Attributes

The term *configuration attributes* means attributes of a configuration produced by the runtime Oracle Configurator. Attributes are predefined data items that record qualities of something. Configuration attributes record data generated during a particular configuration session for qualities that are not defined in the configuration model.

The use of configuration attributes is a methodology for using certain existing features of Oracle Configurator and host applications to capture and exchange data that is not standard inventory information. This kind of data can be especially valuable for

processing by other applications.

The configuration attributes methodology was introduced in a previous release of Oracle Configurator. For more details, examples, and important background on the previous implementation, see *Oracle Configurator Methodologies*.

Important: This chapter only describes adding configuration attributes to an FCE Model that does not already use them. If you are already using configuration attributes with a model that uses the Original Configurator Engine, and you have converted that Model to use the FCE, and you want to use configuration attributes with the new FCE Model, then you will have to change your setup for the FCE Model to accord with the setup described here. See the *Oracle Configurator Release Notes, Release 12.1.1* (On Metalink, Oracle's technical support Web site) for background on these differences. Your configuration attributes setup on your original Model will continue to operate without change.

The setup for configuration attributes consists of the elements described in the table Elements of the Configuration Attributes Setup, page 5-2.

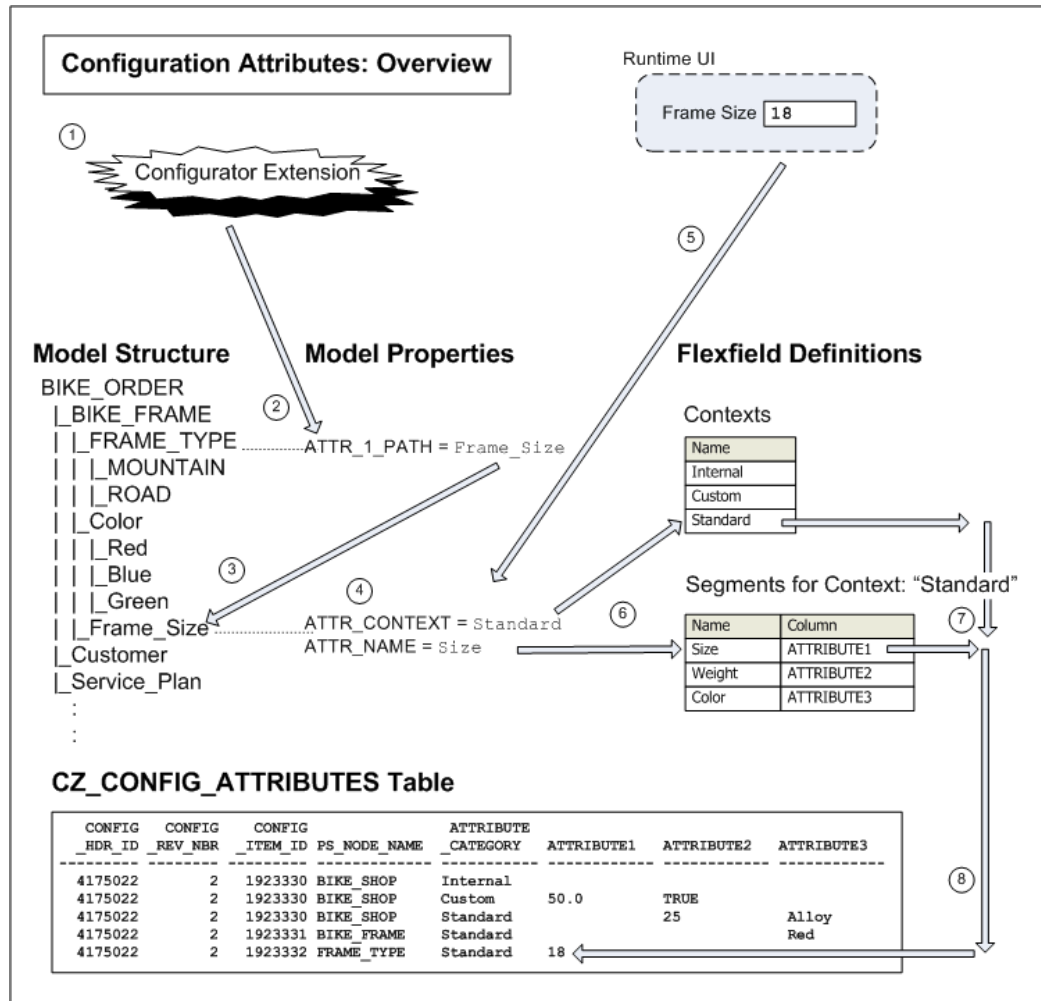
Elements of the Configuration Attributes Setup

Element	Purpose
A configuration model that includes:	
- Imported BOM Model nodes	Configuration attributes describe attributes of these nodes that are not defined in the inventory item.
- Feature nodes	At runtime, values for the configuration attributes are stored in certain specified Features. Features used for this purpose are called <i>attribute Features</i> .
- Properties	Certain specially-named Properties associate BOM Nodes with their corresponding attribute Features. Other specially-named Properties associate the attribute Features with descriptive flexfield segments for a given context. Properties used for this purpose are called <i>attribute Properties</i> .

Element	Purpose
Descriptive flexfield definitions	Flexfield segments connect attribute Features with columns in CZ_CONFIG_ATTRIBUTES, for a given set of flexfield contexts
CZ_CONFIG_ATTRIBUTES	<p>This table stores the values for the configuration attributes so that they can be read for further processing by downstream applications.</p> <p>This table is called the <i>attribute flexfield table</i>.</p>
Configurator Extension	Turns on runtime attribute processing, which gathers the runtime values from the attribute Features of the Model and writes them to the attribute flexfield table.

An example of how a configuration attribute is set up, and how it is used at runtime, is provided in the figure Overview of Configuration Attributes, page 5-4. The flow of the example is explained in the table Configuration Attributes Flow, page 5-4. The details of this setup are explained in following sections, which refer to the elements in this example.

Overview of Configuration Attributes



Configuration Attributes Flow

Step	Process	Example
1	At runtime, when a configuration is initialized (created or restored), the <code>postConfigInit()</code> method of <code>CZAttributeCX</code> is triggered by a Configurator Extension Rule. This method turns on attribute processing, which collects attribute information on the nodes in the configuration model by searching for specially-named attribute Properties.	FRAME_TYPE is the BOM node that is set up for configuration attributes.

Step	Process	Example
2	Attribute processing searches BOM nodes for attribute Properties with names of the form ATTR_#_PATH.	The attribute Property ATTR_1_PATH is found on the BOM node FRAME_TYPE.
3	Attribute processing searches for the node named by ATTR_#_PATH, under the BOM Model node closest to the node	<p>The value of ATTR_1_PATH is Frame_Size, which indicates that there is an attribute Feature named Frame_Size</p> <p>FRAME_TYPE is a BOM Option Class. The closest BOM Model node is BIKE_FRAME.</p> <p>The node named Frame_Size is found under BIKE_FRAME.</p>
4	On the specified attribute Feature, attribute processing obtains the values of the attribute Properties ATTR_NAME and ATTR_CONTEXT.	The value of ATTR_NAME is Size. The value of ATTR_CONTEXT is Standard.
5	An end user enters a value for an attribute Feature.	The user enters the value 18 for Frame_Size.
6	When the configuration is saved, the descriptive flexfield tables are queried using the values of ATTR_NAME and ATTR_CONTEXT.	<p>The value of ATTR_NAME is Size. The value of ATTR_CONTEXT is Standard.</p> <p>These values indicate that the attribute Feature is associated with a descriptive flexfield segment named Size, and a flexfield context of Standard.</p>
7	The values for descriptive flexfield segment and context are used to determine which ATTRIBUTE# column in CZ_CONFIG_ATTRIBUTES is associated with the segment and context.	The flexfield setup indicates that the segment Size, for the context Standard, is associated with column ATTRIBUTE1.
8	The value of the attribute Feature is written to the desired column in CZ_CONFIG_ATTRIBUTES.	The value 18, for the attribute Feature Frame_Size, is written to ATTRIBUTE1.

Tasks for Adding Configuration Attributes to an FCE Model

To set up configuration attributes for an FCE Model, perform the tasks listed in the table

Summary of Setup Tasks for Configuration Attributes, page 5-6. For more information about the tasks, refer to the sections cited in the Details column.

Summary of Setup Tasks for Configuration Attributes

Step	Task	Details
1	Set up descriptive flex field contexts and segments.	Setting Up Descriptive Flexfields, page 5-6
2	Add attribute Features to a BOM Model.	Adding Attribute Features, page 5-8
3	Associate the attribute Features to flexfield segments.	Associating Attribute Features to Flexfield Segments, page 5-10
4	Associate BOM nodes to attribute Features.	Associating BOM Nodes with Attribute Features, page 5-10
5	Define the Configurator Extension Rule.	Defining the Configurator Extension Rule, page 5-12

Important: This chapter only describes adding configuration attributes to an FCE Model. If you are already using configuration attributes with a model that uses the Original Configurator Engine, and you have converted that Model to use the FCE, and you want to use configuration attributes with the new FCE Model, then you will have to change your setup for the FCE Model to accord with the setup described here. See the *Oracle Configurator Release Notes, Release 12.1.1* (On Metalink, Oracle's technical support Web site) for background on these differences. Your configuration attributes setup on your original Model will continue to operate without change.

Setting Up Descriptive Flexfields

The configuration attributes methodology uses the existing Oracle Applications feature of descriptive flexfields to connect the set of attributes on a configuration model with a database table that stores the values of those attributes.

- To define the contexts and segments for the flexfields, log into Oracle Applications with the Oracle Application Developer responsibility.

For conceptual background on descriptive flexfields and contexts, and details on

defining them, see the *Oracle E-Business Suite Flexfields Guide*.

- When you define the contexts for the flexfields, you must create whatever contexts are required by the host application. For all the contexts that you define, use the settings shown in the table Flexfield Settings for All Configuration Attributes Contexts, page 5-7.

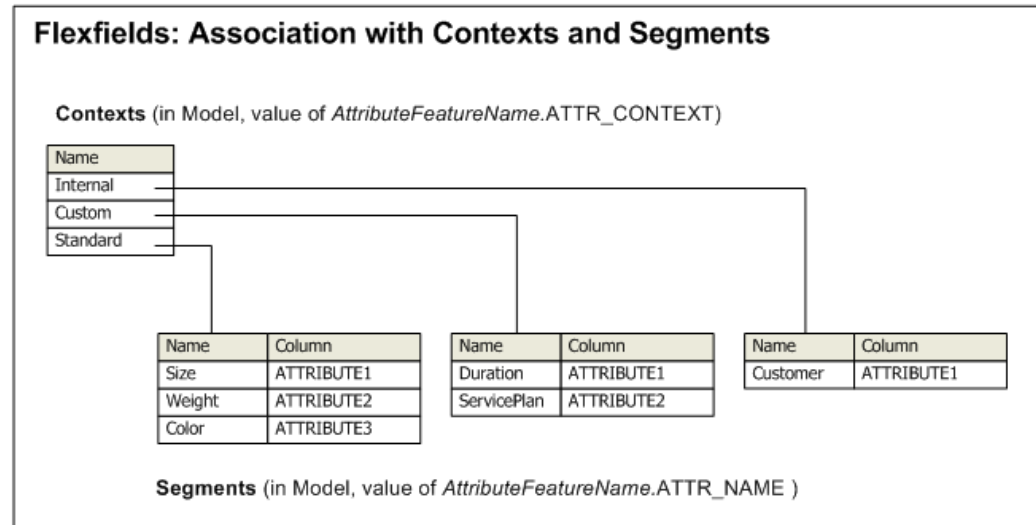
Flexfield Settings for All Configuration Attributes Contexts

Setting	Value
Application	Oracle Configurator
Table Application	Oracle Configurator
Table Name	CZ_CONFIG_ATTRIBUTES

- When you define the segments for each context, you must specify the ATTRIBUTE_n column in CZ_CONFIG_ATTRIBUTES in which that segment's data will be written. By using different contexts you can use the same column to store attribute data for different attribute Features.

The figure Flexfield Contexts and Segments, page 5-8 shows the relationship of flexfield contexts and their segments to the Properties of an attribute Feature. Compare this figure to the figure Association of Attribute Features to Flexfields, page 5-10.

Flexfield Contexts and Segments

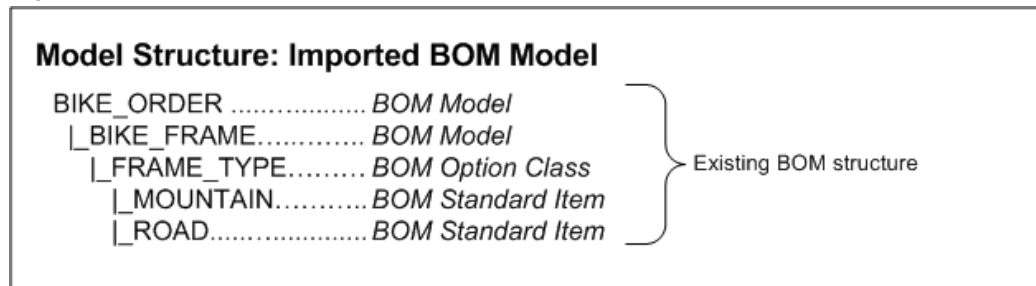


Adding Attribute Features

The structure of a BOM Model does not have a location for storing configuration attributes data. To create such locations, you must add structure to imported BOM Models, in Oracle Configurator Developer. In the configuration attributes methodology, the added structure takes the form of *attribute Features*. Attribute Features are not a new type of Feature. The term attribute Feature simply means a Feature that is used to contain the data for a configuration attribute.

The figure Imported BOM Model, page 5-8 shows the structure of the imported BOM Model for this example, before addition of attribute features.

Imported BOM Model



This imported BOM structure does not have locations for storing certain pieces of data that you might want to capture during a configuration session for ordering a bicycle, such as:

- the color of the BIKE_FRAME being ordered

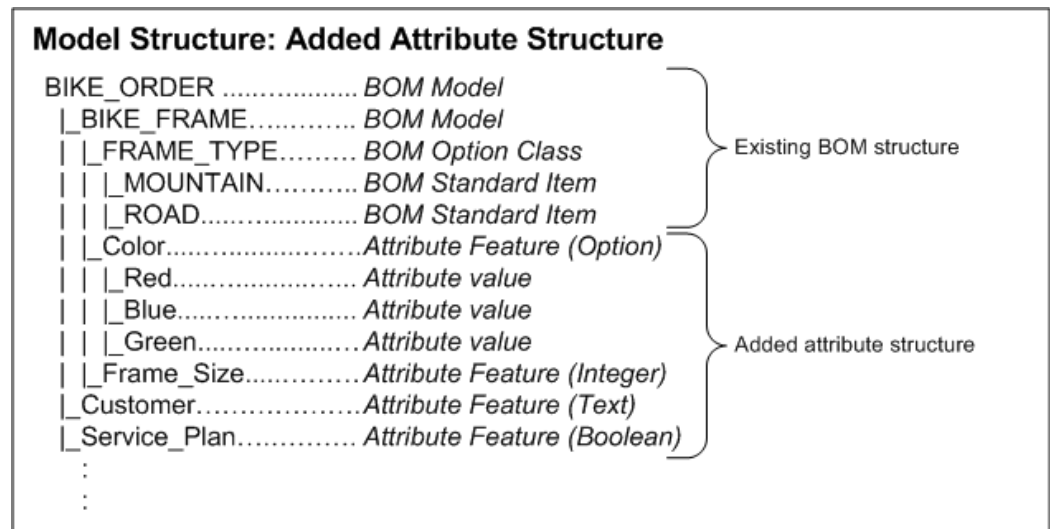
- the frame size of the BIKE_FRAME being ordered
- the name of the customer placing the BIKE_ORDER
- whether that customer elected a service plan for the BIKE_ORDER

To provide locations for entering runtime values for configuration attributes for this Model, create the following Feature nodes:

- **Color:** to capture the color of the BIKE_FRAME being ordered
- **Frame_Size:** to capture the frame size of the BIKE_FRAME being ordered
- **Customer:** to capture the name of the customer placing the BIKE_ORDER
- **Service_Plan:** to capture whether that customer elected a service plan for the BIKE_ORDER

The figure Model with Attribute Features, page 5-9 shows the structure of the modified Model for this example, after the addition of the attribute Features listed in this section. Compare this figure to the figure Imported BOM Model, page 5-8, and to step 5 in the table Configuration Attributes Flow, page 5-4.

Model with Attribute Features



Attribute Features work in conjunction with descriptive flexfield segments (which are described in Setting Up Descriptive Flexfields, page 5-6). The value provided at runtime for an attribute Feature is stored, when the configuration is saved, in the attribute flexfield table CZ_CONFIG_ATTRIBUTES in the column specified by the associated flexfield segment's definition.

For more information about defining Features, see the *Oracle Configurator Developer*

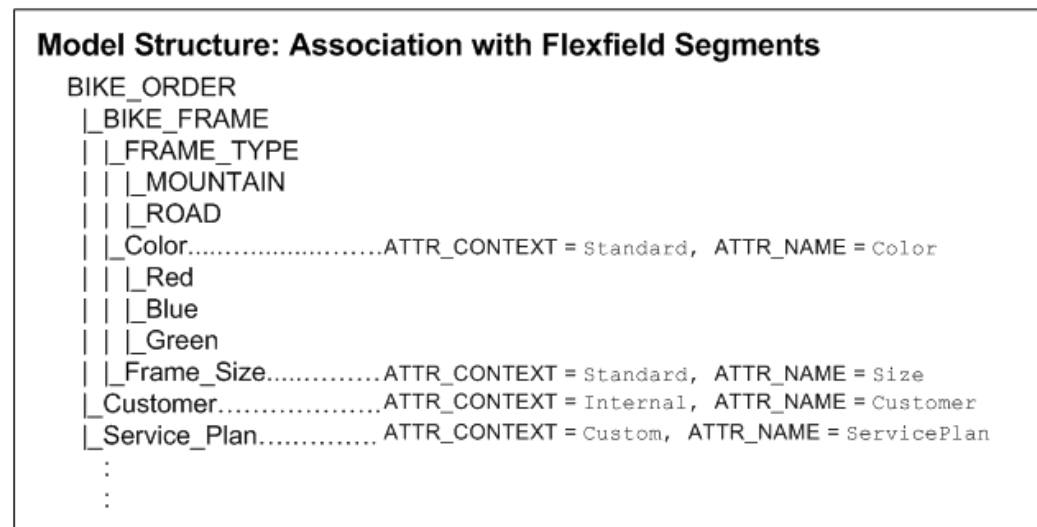
Associating Attribute Features to Flexfield Segments

To create associations between attribute Features and their corresponding descriptive flexfield contexts and segments, you must add attribute Properties to the Features, in Oracle Configurator Developer.

- To associate an attribute Feature to the corresponding descriptive flexfield segment, define a Property of type Text, named ATTR_NAME. The value of the Property must be the name of the corresponding segment.
- To associate an attribute Feature to the corresponding descriptive flexfield context, define a Property of type Text, named ATTR_CONTEXT. The value of the Property must be the name of the corresponding context.

The figure Association of Attribute Features to Flexfields, page 5-10 shows how the attribute Properties ATTR_CONTEXT and ATTR_NAME define associations between attribute Features and their corresponding descriptive flexfield contexts and segments. Compare this figure to the figure Flexfield Contexts and Segments, page 5-8, and to steps 4 and 6 in the table Configuration Attributes Flow, page 5-4.

Association of Attribute Features to Flexfields



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

Associating BOM Nodes with Attribute Features

To create associations between BOM nodes and the attribute Features that store their

configuration attribute values, you must add attribute Properties to the BOM nodes, in Oracle Configurator Developer.

- The Property must be named ATTR_ *n* _PATH, where *n* is an integer that makes the name unique within the scope of the current node.
- The value of the Property must be the node path to the attribute Feature, starting from the BOM Model node that is nearest to the node with the ATTR_ *n* _PATH Property.

If the node with the path property is a BOM Model, then that nearest BOM Model node is itself. If not, then the nearest BOM Model node is found by recursively looking upward in the Model tree until the nearest BOM Model node is found.

- The nodes in the node path are delimited by a dot ('.').
- You can create multiple ATTR_ *n* _PATH Properties for the same node, with different values of *n*, to assign multiple attribute Features to the node. This means that you can assign multiple configuration attributes to the same inventory Item.
- You can use the same ATTR_ *n* _PATH name in different nodes without conflict. (In ordinary practice, you would probably start by using ATTR_1_PATH each time you created a Property on a different node.)
- See steps 2 and 3 in the table Configuration Attributes Flow, page 5-4 for an example of an ATTR_ *n* _PATH Property and how it is used by attribute processing.

The figure Association of BOM Nodes to Attribute Features, page 5-12 shows how the attribute Property ATTR_ *n* _PATH is defined to associate BOM nodes with attribute Features. Compare this figure to steps 2 and 3 in the table Configuration Attributes Flow, page 5-4.

Association of BOM Nodes to Attribute Features

Model Structure: Association with Attribute Features

```
BIKE_ORDER ..... ATTR_1_PATH = Customer, ATTR_2_PATH = Service_Plan
|_BIKE_FRAME..... ATTR_1_PATH = Color
| |_FRAME_TYPE..... ATTR_1_PATH = Frame_Size
| | |_MOUNTAIN
| | |_ROAD
| | |_Color
| | |_Red
| | |_Blue
| | |_Green
| | |_Frame_Size
| |_Customer
| |_Service_Plan
:
:
:
```

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

Defining the Configurator Extension Rule

To provide attribute processing, you must define a Configurator Extension Rule as described in this section.

To ensure that attribute processing is turned on for the entire configuration session, and that processing is performed on all nodes, the event binding for the Rule must specify the Event as `postConfigInit`, which is the first event raised in a session.

The Java class required for this rule, `CZAttributeCX`, is already installed in the class path used by the runtime Oracle Configurator. Consequently, you do not have to perform many of the steps ordinarily required for providing a Java class for a Configurator Extension (compiling Java source code, placing compiled classes in a Java class archive file, creating a Configurator Extension Archive, and adding it to the Model's archive path). Instead, you can just enter its fully-qualified class name in the Rule definition.

Procedure

1. In Oracle Configurator Developer, create a Configurator Extension Rule.
2. Choose the options listed in the following table:

Option	Choice
Model Node	The root node of the Model. (In the example shown here, the root node is BIKE_ORDER.)
Java Class	<code>oracle.apps.cz.cx.CZAttributeCX</code> (Enter this fully-qualified class name directly as shown. Do not select Choose Class.)
Java Class Instantiation	With Model Node Instance

3. Create an event binding for the Configurator Extension Rule, choosing the options listed in the following table:

Option	Choice
Event	<code>postConfigInit</code>
Event Scope	Global (This scope is the only choice for this event.)
Method	<code>postConfigInit (oracle.apps.cz.core.IRuntimeNode)</code>

4. Create an argument binding for the event binding, choosing the options listed in the following table:

Option	Choice
Argument Type	<code>oracle.apps.cz.core.IRuntimeNode</code>
Argument Specification	System Parameter
Binding	<BaseNodeOfRule>

5. Generate logic for your Model, to reflect the addition of the Configurator Extension Rule.

For more information about defining Configurator Extension Rules, see the *Oracle Configurator Developer User's Guide*.

Access to Configuration Attribute Data

The goal of the configuration attributes methodology is to place attribute data in a database table where it can be used by downstream applications. See the following sections for details:

- About the CZ_CONFIG_ATTRIBUTES Table, page 5-14
- Writing Data to the CZ_CONFIG_ATTRIBUTES Table, page 5-15
- Reading Data from the CZ_CONFIG_ATTRIBUTES Table, page 5-15

About the CZ_CONFIG_ATTRIBUTES Table

The attribute flexfield table, CZ_CONFIG_ATTRIBUTES, is used as the intermediate store of configuration attribute data between Oracle Configurator and a host application.

The CZ_CONFIG_ATTRIBUTES Table

Column Name	Null?	PK?	Type	Comments
CONFIG_HDR_ID	N	Y	NUMBER(9)	Configuration header ID
CONFIG_REV_NBR	N	Y	NUMBER(9)	Configuration revision number
CONFIG_ITEM_ID	N	Y	NUMBER(9)	Configuration item ID
ATTRIBUTE_CATEGORY	N	Y	VARCHAR2(30)	Name of flexfield context
ATTRIBUTE1	N	N	VARCHAR2(4000)	Flexfield segment value for the specified context
ATTRIBUTE2	N	N	VARCHAR2(4000)	Flexfield segment value for the specified context
...

Column Name	Null?	PK?	Type	Comments
ATTRIBUTE30	N	N	VARCHAR2(400 0)	Flexfield segment value for the specified context

The columns `CONFIG_HDR_ID`, `CONFIG_REV_NBR`, `CONFIG_ITEM_ID` and `ATTRIBUTE_CATEGORY` constitute the primary key for `CZ_CONFIG_ATTRIBUTES`.

All of the columns `ATTRIBUTE1` through `ATTRIBUTE30` are defined identically, and the columns between `ATTRIBUTE2` and `ATTRIBUTE30` have been omitted from the table `The CZ_CONFIG_ATTRIBUTES Table`, page 5-14 for brevity. The standard columns such as `LAST_UPDATE_DATE` have also been omitted for brevity.

For information about the Oracle Configurator schema, see the *Oracle Configurator Implementation Guide*. For technical details about `CZ_CONFIG_ATTRIBUTES`, `CZ_CONFIG_EXT_ATTRIBUTES`, and other tables, see the Configurator *eTRM* on Metalink, Oracle's technical support Web site.

Writing Data to the CZ_CONFIG_ATTRIBUTES Table

When attribute processing is in effect, and a runtime configuration is saved, then the Core Java API for Oracle Configurator (`oracle.apps.cz.core`) writes the accumulated attribute data to the attribute flexfield table `CZ_CONFIG_ATTRIBUTES`, using the flexfield associations for guidance on which columns to write into.

When a runtime configuration is saved, the API method that is invoked takes a list of BOM nodes that have associated configuration attributes. For each BOM node in the list, the attributes are sorted by flexfield context, which corresponds to the `ATTRIBUTE_CATEGORY` column of the table. Then for each attribute category for that BOM node, a row is written to the table. The row contains all the attribute values for that node and context. The flex field tables (`FND_DESCR_FLEX_COLUMN_USAGES`, `FND_DESCRIPTIVE_FLEXS` and `FND_APPLICATION`) are queried to determine the mapping between attribute-context and flex field column.

Reading Data from the CZ_CONFIG_ATTRIBUTES Table

When you need to obtain configuration attribute data from the `CZ_CONFIG_ATTRIBUTES` table, you can use a SQL query like the one shown in the example `Query for Attribute Data`, page 5-16. The results of this query are shown in the figure `Configuration Attribute Data`, page 5-17.

You must write any custom procedures required to provide configuration attribute data to downstream applications.

To run the query shown here, replace the values for `a.CONFIG_HDR_ID` and `a.CONFIG_REV_NBR` with the Configuration Header ID and Configuration Revision values returned when a configuration session ends and the configuration has been

saved. When you use the Test Model function of Oracle Configurator Developer, these values are displayed on the Confirmation page. When you save a configuration by other means (for instance, from a host application), then the Configuration Header ID and Configuration Revision values are written to the `config_header_id` and `config_rev_nbr` elements of the XML termination message. See the *Oracle Configurator Implementation Guide* for information about the termination message.

Query for Attribute Data

```
SELECT
  a.CONFIG_HDR_ID,
  a.CONFIG_REV_NBR,
  a.CONFIG_ITEM_ID,
  i.PS_NODE_NAME,
  a.ATTRIBUTE_CATEGORY,
  a.ATTRIBUTE1,
  a.ATTRIBUTE2,
  a.ATTRIBUTE3,
  a.ATTRIBUTE10
FROM
  CZ_CONFIG_ATTRIBUTES a, CZ_CONFIG_ITEMS i
WHERE
  a.CONFIG_HDR_ID = 4175522 -- from saved configuration
AND
  a.CONFIG_REV_NBR = 1     -- from saved configuration
AND
  a.CONFIG_ITEM_ID = i.CONFIG_ITEM_ID
AND
  a.CONFIG_HDR_ID = i.CONFIG_HDR_ID
AND
  a.CONFIG_REV_NBR = i.CONFIG_REV_NBR
ORDER BY
  a.CONFIG_ITEM_ID

-- -----
set linesize 400

set headsep on
column CONFIG_HDR_ID heading 'CONFIG|_HDR_ID'
column CONFIG_REV_NBR heading 'CONFIG|_REV_NBR'
column CONFIG_ITEM_ID heading 'CONFIG|_ITEM_ID'
column ATTRIBUTE_CATEGORY heading 'ATTRIBUTE|_CATEGORY'

column PS_NODE_NAME format a16
column ATTRIBUTE_CATEGORY format a24
column ATTRIBUTE1 format a12
column ATTRIBUTE2 format a12
column ATTRIBUTE3 format a12
/
```

The output in the figure Configuration Attribute Data, page 5-17 shows the result of running the query in the example Query for Attribute Data, page 5-16.

Configuration Attribute Data

Reading Data from the CZ_CONFIG_ATTRIBUTES Table

CZ_CONFIG_ATTRIBUTES Table

CONFIG_HDR_ID	CONFIG_REV_NBR	CONFIG_ITEM_ID	PS_NODE_NAME	ATTRIBUTE_CATEGORY	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3
4175522	1	1923850	BIKE_SHOP	Internal			
4175522	1	1923850	BIKE_SHOP	Custom	0.0	FALSE	
4175522	1	1923850	BIKE_SHOP	Standard		20	Alloy
4175522	1	1923851	BIKE_FRAME	Standard			Red
4175522	1	1923852	FRAME_TYPE	Standard	18		

Runtime value for:
Attribute Feature: Frame_Size
Where:
Flexfield Context: Standard
Flexfield Segment: Size

Special Considerations

This section covers some important considerations that may affect your implementation of the configuration attributes methodology.

- **Referenced Models**

If your Model includes child Models that are connected through References (as is the case with most imported BOM Models), then attribute processing traverses the entire structure of the Model, collecting configuration attribute data from any attribute Properties whose names conform to the conventions described in this chapter.

- **Location of Attribute Features**

You must give careful thought to where you create attribute Features. An attribute Feature can be located anywhere in a Model, provided that the location is a node path that can be specified by the Property ATTR_n_PATH. Consequently, the node path to an attribute Feature must be specified as relative to the Model that contains the Item.

Attribute processing first finds the node that is the root of the entire configuration model, and then searches from there for Properties whose names conform to the conventions described in this chapter.

Because attribute processing can find Features in child Models that are connected through References, it is possible to assign a configuration attribute value to an Item in a parent Model by pointing (with the Property ATTR_n_PATH) to an attribute Feature in a referenced child Model. Such a node path must include the names of any child Models between the Model that contains the Item and the attribute Feature. However, a node path that you can express as a Text Property value can

only point down the tree of Models, not upwards. Consequently, it is not possible to point to an attribute Feature in a parent Model, and therefore you cannot assign a configuration attribute value from a parent Model to an Item in a child Model.

Here is an illustration, using the example Model cited elsewhere in this chapter and shown in the figure Association of BOM Nodes to Attribute Features, page 5-12.

When the configuration attribute is an attribute (Size) of a BOM Option Class (FRAME_TYPE), then the attribute Feature (Frame_Size) must be a child of that BOM Option Class's parent model (BIKE_FRAME), which is the nearest BOM Model node. The path from that BOM Model (from BIKE_FRAME to Frame_Size) cannot be up, only down (as it is). The node path generally cannot traverse child references, unless the reference is a required single instance.

To summarize: parent Models can use attribute Features defined in referenced child Models; child Models cannot use attribute Features defined in parent Models.

It follows from the preceding facts that if you intend to use a referenced child Model without the parent Model, then you must ensure that it contains all the attribute Features that are employed as the configuration attributes on the Items in that Model.

- **Multiple Component Instances in the Node Path**

The node path to an attribute Feature, which is specified by the Property ATTR_n_PATH, cannot include any Components that can be instantiated multiple times. (Such a Component is one that has its Instances set to a Maximum greater than 1, so that more than one runtime instance of the Component can be created and configured.) The existence of multiple instances of a Component in a node path makes the path ambiguous. Consequently, you cannot place any attribute Features inside such a Component, because attribute processing cannot resolve the correct path to that attribute Feature. This restriction also applies to Components whose instantiability is Optional Single Instance.

- **Reusing an Attribute Value for Multiple Items**

It may be required that more than one Item needs the same configuration attribute value. You can accomplish this by making the Items point to the same instance of the attribute Feature that contains that value. You do this by setting the same value for the attribute Property ATTR_n_PATH in each Item.

- **Required Items**

Configuration attributes cannot be defined for required Standard Items in a BOM Model, because such Items are not configurable, and consequently are not imported when you import the BOM Model into Oracle Configurator Developer to define a configuration model. (See the *Oracle Configurator Implementation Guide* for details about importing.)

- **Effects of Auto-Complete and Adjust Mode**

After a successful Auto-Complete operation, any values for attribute Features that were bound by the results of Auto-Complete are converted to Auto-Complete Decisions (ACDs). (ACDs keep track of the selections made by Auto-Complete itself. ACDs are retained during Adjust Mode, and are re-applied when the user subsequently chooses Finish or Auto-Complete.) This functionality enables configuration attributes to capture values for all attribute Features that you consider to be output of the configuration, not only those explicitly selected by the end user.

Maintaining the Configuration Attributes Setup

You must manually update your Model in Oracle Configurator Developer if the following are changed:

- The flexfield definitions in Oracle Applications. Changes to the flexfield definitions might include changes to context or attribute column assignments.
- The BOM (in Oracle Bills of Materials) that is the basis for your imported Model. Changes to the Bill of Material might include the addition of an Item that should have configuration attributes.

If the BOM changes, you can reflect the changes by refreshing your Model. See the *Oracle Configurator Implementation Guide* for details on refreshing Models. If you refresh your Model, you may need to make changes in your attribute Features and Properties.

Using Configuration Attributes in the Downstream Application

For suggestions about using configuration attributes in a downstream application, see the following sections in *Oracle Configurator Methodologies*:

- "Using Configuration Attributes in the Downstream Application"
- "Using Output Data in Downstream Applications"
- "Linking Configuration Attributes to Flexfields"
- "Downstream User Interfaces"

CIO Emulation for the FCE

This chapter covers the following topics:

- About CIO Emulation for the FCE
- Tasks for Implementing CIO Emulation for the FCE
- Converting Source Files with Substitution
- Compiling and Archiving Converted Files
- Converting Configurator Extension Rules
- Verifying Post-Conversion Behavior

About CIO Emulation for the FCE

CIO Emulation for the FCE is a facility that assists you in modifying existing Configurator Extensions that were written for the Original Configurator Engine (OCE) so that they might be able to operate with the Fusion Configurator Engine (FCE).

Why CIO Emulation for the FCE Is Needed

When you convert a Model to use the FCE, as described in *Converting Existing Models to Use the Fusion Configurator Engine*, page 2-7, the Model Conversion Utility does not convert the Configurator Extensions associated with that Model so that they also use the FCE.

Configurator Extensions that are associated with OCE Models interact with the Model through the Java API called the Configuration Interface Object (CIO), `oracle.apps.cz.cio`. The CIO is tailored specifically to the OCE. The FCE is entirely different in architecture from the OCE, and a different API is provided for interacting with FCE Models. This new API for the FCE is the Core API, `oracle.apps.cz.core`, which is documented by the Oracle Configurator API Reference, available through My Oracle Support.

The CIO API cannot access FCE Models. Therefore, CIO-based Configurator Extensions

associated with an OCE Model will not work when the Model is converted to an FCE Model. Consequently, you must consider what to do with your Configurator Extensions. The options are described in the table Options for CIO-Based Configurator Extensions, page 6-2.

Options for CIO-Based Configurator Extensions

Option	Advantages	Disadvantages
Rewrite CX to use the Core API.	Takes advantage of FCE capabilities, and provides optimal functionality.	Significant development effort. Requires Java expertise. Not necessary, if CX functionality is duplicated by FCE.
Use CIO Emulation for the FCE .	Requires potentially small amount of work, since conversion is mostly automated. Potentially requires less Java expertise.	Functionality of CX after conversion must be verified. Differences in behavior may require new programming. May require some Java work to resolve compilation errors and other incompatibilities.
Discontinue use of CX.	If CX functionality is duplicated by FCE, no loss by discontinuing CX.	If CX functionality is not duplicated by FCE, effect of CX is lost.

Intended Audience for CIO Emulation for the FCE

CIO Emulation for the FCE was designed primarily to provide a relatively straightforward migration path to the FCE for developers who have written Configurator Extensions that perform relatively straightforward tasks. If you have used the CIO to build a large-scale custom application, then the CIO Emulation for the FCE is not likely to help you migrate to the FCE, and you should consider rewriting your code using the Core API.

Elements of CIO Emulation for the FCE

The following elements constitute the CIO Emulation for the FCE facility:

- **CIO Emulation API** (`oracle.apps.cz.cioemu`)

This API provides classes and methods that emulate the syntax of the CIO, while

indirectly accessing the Core API. The API is documented by the Oracle Configurator API Reference, available through My Oracle Support. The relationship of this API to the CIO and Core APIs is shown in the figure APIs Related to CIO Emulation, page 6-4.

- **Substitution Script**

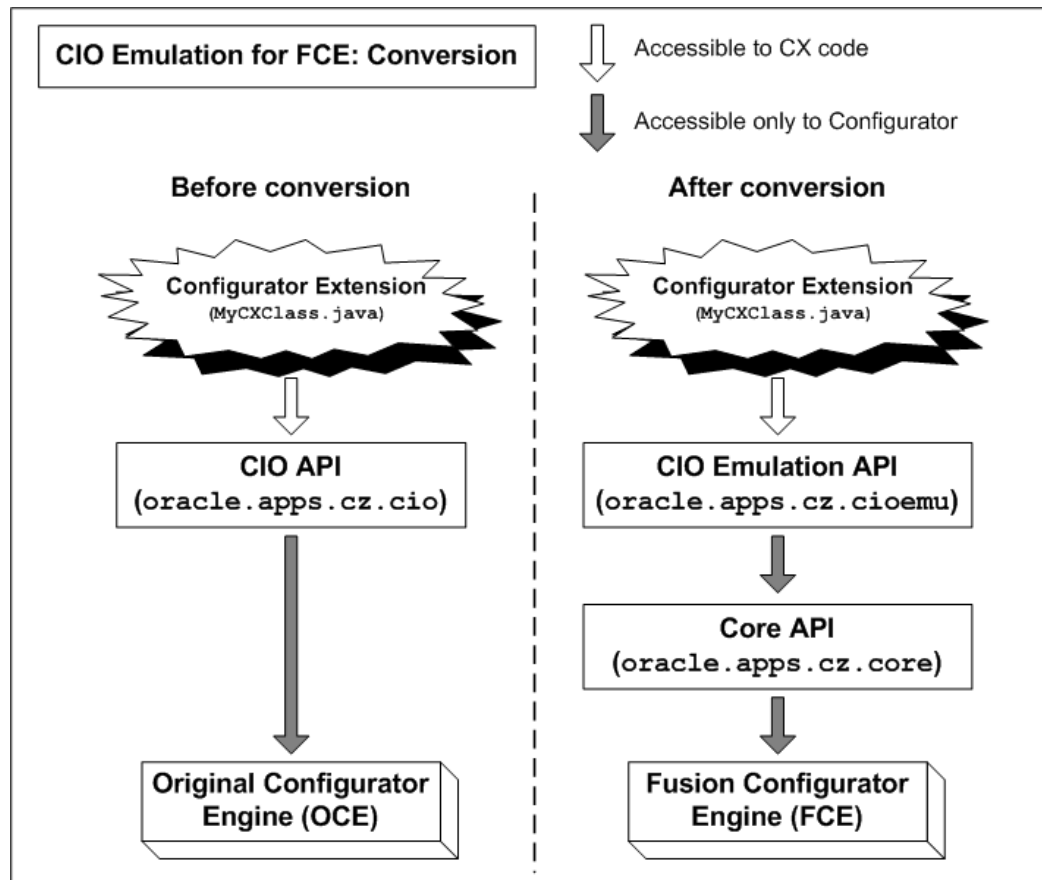
This script modifies the source code of your Configurator Extensions, changing certain identifiers to refer to the CIO Emulation API rather than to the CIO API. For details, see Converting Source Files with Substitution, page 6-11.

- **Configurator Extension Status**

This modification to Oracle Configurator Developer enables you to convert existing Configurator Extension Rules so that they refer to the CIO Emulation API rather than to the CIO API. For details, see Converting Configurator Extension Rules, page 6-23.

For an overview of the process that these elements support, see Tasks for Implementing CIO Emulation for the FCE, page 6-10.

APIs Related to CIO Emulation



Differences Between FCE and CIO Emulation

This section describes features how differences in behavior between the Fusion Configurator Engine and the Original Configurator Engine are handled in the CIO Emulation API, where possible.

Behavior Differences Between FCE and CIO Emulation

CIO (or FCE) Feature	FCE Behavior	CIO Emulation Behavior
Configuration transactions (<code>ConfigTransaction</code>)	No true equivalent for configuration transactions. The FCE has the concept of <code>SavePoints</code> .	<p>This feature is not supported</p> <p>A warning is provided by the substitution script.</p> <p>The <code>Configuration</code> object provides "no-operation" methods that allow compilation but produce no results.</p> <p>There is no support for transaction levels.</p>
<code>StatusInfo</code>	No equivalent concept in the Core API.	<p>This feature is not supported.</p> <p>Any use of <code>StatusInfo</code> will not compile.</p>
<code>RestoreValidationFailure</code>	The Core API has different handling of failed inputs when restoring a configuration, generating informational messages instead of validation failures.	<p>This feature is not supported.</p> <p>A warning is provided by the substitution script.</p> <p>An emulation object is provided for compilation, but the Core API will not create objects, and CX code using this class will never be executed.</p>
TSO (MACD)	The Telecommunications Services Ordering (TSO) functionality is not available for the Fusion Configurator Engine.	<p>This feature is not supported.</p> <p>Since TSO functionality is not available for the FCE, there is no need for CIO Emulation support for TSO.</p> <p>Attempts to compile converted CX code using TSO features will raise compiler errors.</p>

CIO (or FCE) Feature	FCE Behavior	CIO Emulation Behavior
Configuration attributes (TSO-based)	A new implementation of configuration attributes for the FCE is provided, and described elsewhere in this document.	TSO-based configuration attributes are not supported. Attempts to compile CX code using TSO-based attribute features will raise compiler errors
Total and Resource objects	These OCE objects are converted to <code>DecimalValue</code> . During upgrade, domain definitions will be set as needed.	Replaced by <code>cioemu.Total</code> and <code>cioemu.Resource</code> objects.
Unbound Values	If a value is requested before being bound, Core API throws exception.	All Emulation getter methods will catch <code>NodeUnboundException</code> and provide a value consistent with the CIO behavior. Value is <code>Unknown</code> for state nodes and <code>0</code> for numeric nodes.
Logical exceptions	The FCE supports Logical Exceptions, as the CIO does, but the implementation is different.	Supported with emulation Logical Exceptions, which provide equivalent behavior.
CIO Exceptions (such as <code>InsufficientCountException</code>)	The Core API (for the FCE) does not throw all the same exceptions as the CIO API (for the OCE). CIO exceptions not needed by the Core API are not supported.	Emulation versions of these types of exceptions are provided. Any converted CX code using these exceptions will compile but the code will never be executed since the Core API does not throw these types of exceptions.

CIO (or FCE) Feature	FCE Behavior	CIO Emulation Behavior
<p>Connectors</p> <p>Note: Connectors are not yet supported for the FCE. This information is provided for planning.</p>	A <code>ConnectorPort</code> can have multiple connections.	Supported. But if cardinality of connections is greater than 1, then throws an exception when trying to create the emulation object, since OCE models can only have 1 target per connection. If there is more than 1, then either the conversion has an issue or the model has been changed.
Editable Totals and Resource	Not supported.	<p>Not supported.</p> <p>Any calls in code to set the value of a Total or Resource will not compile.</p>
<p>FCE Proposed True and Proposed False states</p> <p>(Recommended (Proposed Selected) and Not Recommended (Proposed Excluded).</p>	These are new values returned from calls with <code>getState()</code> . Customer code not using existing static CIO helper methods (such as <code>isTrue(int state)</code>) may not behave as expected.	Since OCE doesn't have the concept of PTRUE the emulation layer will return PTRUE as LTRUE and PFALSE as LFALSE.
InformationalMessage	There is no equivalent object in the Core API. Informational messages are now text-based.	The CIO Emulation API provides an equivalent object for <code>InformationalMessage</code> .
ValidationFailure	The Core API never creates <code>ValidationFailure</code> objects as the CIO does. The Core API only supports custom validation failures created by user-written CXs.	The CIO Emulation API provides an equivalent object for <code>ValidationFailure</code> .
CustomValidationFailure	There is no equivalent object in the Core API.	The CIO Emulation API provides an equivalent object for <code>CustomValidationFailure</code> , extending <code>cioemu.ValidationFailure</code> .

CIO (or FCE) Feature	FCE Behavior	CIO Emulation Behavior
Intermediate Values	Problematic intermediate values do not arise with the FCE, so this concept is not supported in the Core API. In the Core API, methods representing the value of a numeric feature return zero until the value is bound.	Problematic intermediate values can arise in the OCE, so in the CIO methods representing the value of a numeric feature may return intermediate values. Because of the fundamental difference between OCE and FCE handling of intermediate values, no emulation support can be provided to bridge the gap between them.

Candidate Implementations for Rewriting

This section suggests some Configurator Extension implementations that may be better served by rewriting the CXs than by converting them to use CIO Emulation for the FCE, since emulation does not completely provide the desired functionality.

Most of these CX implementations were intended to remedy shortcomings in the Original Configurator Engine that are described in Original Configurator Engine: Known Issues and Limitations, page 1-6. The *Oracle Configurator Modeling Guide* provides more detail on several of the issues related to the Original Configurator Engine.

It may be better to rewrite the following types of Configurator Extensions:

- CXs that implement non-trivial configuration transactions.
The FCE does not use configuration transactions. Emulation provides "no-operation" transaction methods that allow the emulated CX code to compile and execute, but produce no results.
- CXs that work around limitations in using the NotTrue logical operator or logical function.
These limitations do not occur with the FCE, because items in the configuration can never be in an "Unknown" state.
- CXs that work around Numeric Rules being unidirectional.
In an FCE Model, Numeric constraints propagate in both directions.
- CXs that work around intermediate numeric values in Comparison Rules.
This limitation does not occur with the FCE, due to the way in which it propagates

rules.

- CXs that work around Default limitations.

These limitations do not occur with the FCE, because the FCE implements Defaults in a significantly different way than the OCE.

- CXs that implement auto-instantiation.

These actions are unnecessary with the FCE, because the FCE is able to dynamically instantiate components to satisfy constraints.

- CXs that implement a Search.

This implementation is unnecessary with the FCE, because the Auto-Complete process of the FCE searches for a solution to a configuration.

Limitations of CIO Emulation for the FCE

Keep in mind the following limitations:

- Emulation cannot fulfill every application of the CIO API for Configurator Extensions, since the CIO Emulation API cannot completely emulate all CIO behavior in the FCE. The FCE has capabilities that surpass the CIO, including some that overcome limitations of the CIO. In such cases, you should consider rewriting your Configurator Extensions and redesigning your configuration model to take advantage of the FCE. Keep in mind that the FCE makes many uses of Configurator Extensions unnecessary. See Candidate Implementations for Rewriting, page 6-8 for suggestions.
- You should evaluate the expected lifetime of the Configurator Extensions created by converting them to use CIO Emulation for the FCE, because methods in the CIO Emulation API cannot directly access the Core API. Therefore, you cannot add Core API methods into Configurator Extensions created by the conversion process. This means that you cannot combine OCE functionality and FCE functionality together in a single Configurator Extension. You can, however create new CXs that use only the CIO Emulation API, or new CXs that use only the Core API.

Caution: The use of objects from `oracle.apps.cz.cioemu` and `oracle.apps.cz.core` in the same method is explicitly unsupported. Such methods may compile, but will fail at runtime.

- Before deploying Configurator Extensions using CIO Emulation for the FCE, it is essential to verify their behavior against your expectations, since there may be differences related to the basic differences between the Original Configurator Engine and the Fusion Configurator Engine.

- Due to the limitations described in this chapter, it may not be possible to successfully convert all of your Configurator Extensions.

Tasks for Implementing CIO Emulation for the FCE

To implement CIO Emulation for the FCE, perform the tasks listed in Table Summary of Tasks for Using CIO Emulation for the FCE, page 6-10. For information about the tasks, refer to the sections cited in the Details column.

Summary of Tasks for Using CIO Emulation for the FCE

Task	Description	For Details
1	Run the substitution script on the source files for your package.	Converting Source Files with Substitution, page 6-11
2	Compile the converted source files, resolve compilation errors, then create a Java archive file for the converted classes.	Compiling and Archiving Converted Files, page 6-22
3	Create a Configurator Extension Archive for the Java archive, then use Oracle Configurator Developer to convert existing Configurator Extension Rules that use the converted classes.	Converting Configurator Extension Rules, page 6-23
4	Verify the behavior of your Model when using the converted classes.	Verifying Post-Conversion Behavior, page 6-26

Requirements for Implementing CIO Emulation

The following are required to perform the procedures described in this chapter:

- Linux operating system, or other OS with support for the Perl scripting language
- Perl scripting language, as normally provided in Oracle installations
- It is assumed that your source file directory structure mirrors the structure of your Java packages.

Converting Source Files with Substitution

Before you can use CIO Emulation for the FCE in your Configurator Extensions, you must modify your Java classes to refer to the CIO Emulation API. To assist you in doing this, Oracle provides a substitution script that performs certain modifications on your Java source files.

Note: Note: As described elsewhere in this section, the script is named `ciocxemu.pl`, and it is installed by default, in `$APPL_TOP/cz/12.0.0/bin`.

What the Substitution Script Does

The substitution script performs a set of simple text substitutions on all the Java source files in your current working directory. Subdirectories are never processed.

Important: If you have multiple packages that reference classes in each other, then you should use custom substitution. See Custom Substitution, page 6-18 for details.

The substitution script performs the text substitutions described in the table *Substitutions Performed by the Script*, page 6-11.

Substitutions Performed by the Script

Location in Code	Substitution Performed by the Script
import declarations	Changes all references to <code>oracle.apps.cz.cio</code> to <code>oracle.apps.cz.cioemu</code>
package declarations	Changes the package name of the class defined in the file, adding the suffix <code>.emu</code> . For example, <code>my.pkg.cx</code> is changed to <code>my.pkg.cx.emu</code>
variable declarations, parameter declarations, and type casts	Changes fully qualified class, interface, and exception names as shown in the table <i>Substitutions Performed on the Supported Set of CIO Objects</i> , page 6-12

Location in Code	Substitution Performed by the Script
code comments	Substitutions are applied in comment text. The conversion does not distinguish comments from code.

Substitutions Performed on the Supported Set of CIO Objects

CIO Object Name	Substituted Object Name
oracle.apps.cz.cio.AtpUnavailableException	oracle.apps.cz.cioemu.AtpUnavailableException
oracle.apps.cz.cio.BomExplosionException	oracle.apps.cz.cioemu.BomExplosionException
oracle.apps.cz.cio.BomInstance	oracle.apps.cz.cioemu.BomInstance
oracle.apps.cz.cio.BomModel	oracle.apps.cz.cioemu.BomModel
oracle.apps.cz.cio.BomNode	oracle.apps.cz.cioemu.BomNode
oracle.apps.cz.cio.BomOptionClass	oracle.apps.cz.cioemu.BomOptionClass
oracle.apps.cz.cio.BomStdItem	oracle.apps.cz.cioemu.BomStdItem
oracle.apps.cz.cio.BooleanFeature	oracle.apps.cz.cioemu.BooleanFeature
oracle.apps.cz.cio.Component	oracle.apps.cz.cioemu.Component
oracle.apps.cz.cio.ComponentNode	oracle.apps.cz.cioemu.ComponentNode
oracle.apps.cz.cio.ConfigParameters	oracle.apps.cz.cioemu.IConfigParameters
oracle.apps.cz.cio.Connector	oracle.apps.cz.cioemu.Connector
oracle.apps.cz.cio.ConnectorInfo	oracle.apps.cz.cioemu.ConnectorInfo

CIO Object Name	Substituted Object Name
oracle.apps.cz.cio.ConfigurationMessage	oracle.apps.cz.cioemu.ConfigurationMessage
oracle.apps.cz.cio.CountFeature	oracle.apps.cz.cioemu.CountFeature
oracle.apps.cz.cio.CustomValidationFailure	oracle.apps.cz.cioemu.CustomValidationFailure
oracle.apps.cz.cio.DecimalFeature	oracle.apps.cz.cioemu.DecimalFeature
oracle.apps.cz.cio.DecimalNode	oracle.apps.cz.cioemu.DecimalNode
oracle.apps.cz.cio.IAtp	oracle.apps.cz.cioemu.IAtp
oracle.apps.cz.cio.IBasicPrice	oracle.apps.cz.cioemu.IBasicPrice
oracle.apps.cz.cio.IBomItem	oracle.apps.cz.cioemu.IBomItem
oracle.apps.cz.cio.ICount	oracle.apps.cz.cioemu.ICount
oracle.apps.cz.cio.IDecimal	oracle.apps.cz.cioemu.IDecimal
oracle.apps.cz.cio.IDecimalMinMax	oracle.apps.cz.cioemu.IDecimalMinMax
oracle.apps.cz.cio.IInteger	oracle.apps.cz.cioemu.IInteger
oracle.apps.cz.cio.IIntegerMinMax	oracle.apps.cz.cioemu.IIntegerMinMax
oracle.apps.cz.cio.InformationalMessage	oracle.apps.cz.cioemu.InformationalMessage
oracle.apps.cz.cio.InsufficientCountException	oracle.apps.cz.cioemu.InsufficientCountException
oracle.apps.cz.cio.IntegerFeature	oracle.apps.cz.cioemu.IntegerFeature
oracle.apps.cz.cio.IntegerNode	oracle.apps.cz.cioemu.IntegerNode

CIO Object Name	Substituted Object Name
oracle.apps.cz.cio.IOption	oracle.apps.cz.cioemu.IOption
oracle.apps.cz.cio.IOptionFeature	oracle.apps.cz.cioemu.IOptionFeature
oracle.apps.cz.cio.IPrice	oracle.apps.cz.cioemu.IPrice
oracle.apps.cz.cio.IReadOnlyDecimal	oracle.apps.cz.cioemu.IReadOnlyDecimal
oracle.apps.cz.cio.IRuntimeNode	oracle.apps.cz.cioemu.IRuntimeNode
oracle.apps.cz.cio.IState	oracle.apps.cz.cioemu.IState
oracle.apps.cz.cio.IText	oracle.apps.cz.cioemu.IText
oracle.apps.cz.cio.ModelLookupException	oracle.apps.cz.cioemu.ModelLookupException
oracle.apps.cz.cio.ModifiedConfigOverrideException	oracle.apps.cz.cioemu.ModifiedConfigOverrideException
oracle.apps.cz.cio.NeighborhoodBoundaryException	oracle.apps.cz.cioemu.NeighborhoodBoundaryException
oracle.apps.cz.cio.NoAtpCalculatedException	oracle.apps.cz.cioemu.NoAtpCalculatedException
oracle.apps.cz.cio.NoConfigHeaderException	oracle.apps.cz.cioemu.NoConfigHeaderException
oracle.apps.cz.cio.Option	oracle.apps.cz.cioemu.Option
oracle.apps.cz.cio.OptionFeature	oracle.apps.cz.cioemu.OptionFeature
oracle.apps.cz.cio.OptionFeatureNode	oracle.apps.cz.cioemu.OptionFeatureNode
oracle.apps.cz.cio.OptionNode	oracle.apps.cz.cioemu.OptionNode
oracle.apps.cz.cio.PricedNode	oracle.apps.cz.cioemu.PricedNode

CIO Object Name	Substituted Object Name
oracle.apps.cz.cio.PricingUnavailableException	oracle.apps.cz.cioemu.PricingUnavailableException
oracle.apps.cz.cio.Property	oracle.apps.cz.cioemu.IProperty
oracle.apps.cz.cio.PropertyNotAvailableException	oracle.apps.cz.cioemu.PropertyNotAvailableException
oracle.apps.cz.cio.Reason	oracle.apps.cz.cioemu.Reason
oracle.apps.cz.cio.Request	oracle.apps.cz.cioemu.Request
oracle.apps.cz.cio.Resource	oracle.apps.cz.cioemu.Resource
oracle.apps.cz.cio.RuntimeNode	oracle.apps.cz.cioemu.RuntimeNode
oracle.apps.cz.cio.StateCountNode	oracle.apps.cz.cioemu.StateCountNode
oracle.apps.cz.cio.StateNode	oracle.apps.cz.cioemu.StateNode
oracle.apps.cz.cio.TargetInfo	oracle.apps.cz.cioemu.TargetInfo
oracle.apps.cz.cio.TextFeature	oracle.apps.cz.cioemu.TextFeature
oracle.apps.cz.cio.TextNode	oracle.apps.cz.cioemu.TextNode
oracle.apps.cz.cio.Total	oracle.apps.cz.cioemu.Total
oracle.apps.cz.cio.ValidationFailure	oracle.apps.cz.cioemu.ValidationFailure

The substitution script only changes class and package names. It does not attempt to reimplement your CIO-based code in terms of the API for the FCE. See Differences Between FCE and CIO Emulation, page 6-4.

Running the Script

Procedure

To run the substitution script:

1. Identify the Java source files for the package that you want to convert to use the

CIO Emulation API. It is assumed that your source file directory structure mirrors the structure of your Java packages.

For example, the source files for a package named `my.pkg.cx` would be located in a directory named `/mydev/pkg/cx`.

You can only process a single package at a time. You cannot run the script recursively over subdirectories.

2. Change to the directory containing the files that you want to convert.

```
cd /mydev/pkg/cx
```

3. Enter the command to run the substitution script. See Syntax and Parameters, page 6-16 for information on the syntax of the command and its parameters.

```
perl yourpath/ciouxemu.pl parameters
```

Example command for the package mentioned in step 1 (`my.pkg.cx`):

```
perl $APPL_TOP/cz/12.0.0/bin/ciouxemu.pl my.pkg.cx
```

By default, the script is installed in `$APPL_TOP/cz/12.0.0/bin`.

4. The script processes all the Java source files in the current working directory (but not in any subdirectories), performing substitutions on each line of each file where a substitution is needed. See What the Substitution Script Does, page 6-11.
5. The script writes a series of informational messages to the console. For details on these messages, see Output of the Script, page 6-20.
6. The script creates a new subdirectory of the current working directory, named `emu`. This directory contains the modified Java files produced by the script, and a log file that contains the informational messages that were written to the console by the script. The code in the Java files has been modified as described in What the Substitution Script Does, page 6-11. Your original Java files in the current working directory are left unchanged.
7. Proceed to the next step under Tasks for Implementing CIO Emulation for the FCE, page 6-10.

Syntax and Parameters

The syntax for running the substitution script is:

```
[perl] [yourpath]ciouxemu.pl [-nologfile] [-noscreen]  
original_package [custom_targetcustom_replacement]  
[perl] [yourpath]ciouxemu.pl [ -help | -manual ]
```

The parameters and command-line options for the substitution script are described in the table Options and Parameters for the Substitution Script, page 6-17.

Options and Parameters for the Substitution Script

Option or Parameter	Description
<code>perl</code>	If necessary, invocation of the <code>perl</code> processor. Details are dependent on site and operating system.
<code>yourpath</code>	If necessary, your path to the substitution script <code>ciocxemu.pl</code> . By default, the script is installed in <code>\$APPL_TOP/cz/12.0.0/bin</code> .
<code>-nologfile</code>	Suppresses generation of the log file.
<code>-noscreen</code>	Suppresses display of conversion messages to the console.
<code>original_package</code>	(Mandatory.) The package named in the <code>package</code> declaration in the Java source files being processed.
<code>custom_target</code>	When using custom substitution, the string in the original files to be replaced by <code>custom_replacement</code> in the modified Java files produced by the script. See Custom Substitution, page 6-18.
<code>custom_replacement</code>	When using custom substitution, the string to replace <code>custom_replacement</code> . See Custom Substitution, page 6-18.
<code>-help</code>	Displays basic command information about the script, in plain text format.
<code>-manual</code>	Displays basic command information about the script, in man page format.

The common form of the command for running the script is shown in the example Simple Substitution Syntax, page 6-17. The example command shown in the example Simple Substitution Example, page 6-17 makes the changes shown in the table File Modifications Performed by Simple Substitution, page 6-18.

Simple Substitution Syntax

```
ciocxemu.pl original_package
```

Simple Substitution Example

```
ciocxemu.pl my.pkg.cx
```

File Modifications Performed by Simple Substitution

Original File	Modified File
<pre>package my.pkg.cx;</pre>	<pre>package my.pkg.cx.emu;</pre>
<pre>import oracle.apps.cz.cio .IRuntimeNode;</pre>	<pre>import oracle.apps.cz.cioemu .IRuntimeNode;</pre>
<pre>postCXInit(java.lang.String, oracle.apps.cz.cio.IRuntimeNode)</pre>	<pre>postCXInit(java.lang.String, oracle.apps.cz.cioemu .IRuntimeNode)</pre>

Custom Substitution

Custom substitution allows you to make arbitrary substitutions on the package being processed. It replaces a specified string in your original Java source files with a specified replacement string that will appear in your modified files.

Custom Substitution Syntax

```
ciocxemu.pl original_package custom_target custom_replacement
```

Custom Substitution Example

```
ciocxemu.pl my.pkg.cx my.pkg.util my.pkg.util.emu
```

Custom substitution is especially useful for the common situation in which a package references classes in another package. Consider an example in which one of your classes (`my.pkg.cx.MyCXClass`) calls a method (`formatDate()`) in another package (`my.pkg.util.Toolkit`). The table Cross-Package References with Simple Substitution, page 6-18 shows how simple substitution can result in references that point to the incorrect package and class. The table Cross-Package References with Custom Substitution, page 6-19 shows how custom substitution helps to modify the referencing package to correct many of the references.

Cross-Package References with Simple Substitution

Stage	Class in Referenced Package	Class in Referencing Package
	<code>my.pkg.util.Toolkit</code>	<code>my.pkg.cx.MyCXClass</code>

Stage	Class in Referenced Package	Class in Referencing Package
Before simple substitution	<pre>package my.pkg.util; import oracle.apps.cz.cio.IRuntimeNode; public class Toolkit { void formatDate() }</pre>	<pre>package my.pkg.cx; import oracle.apps.cz.cio.IRuntimeNode; import my.pkg.util.Toolkit; d = my.pkg.util.Toolkit.formatDate();</pre>
Simple substitution on referenced package	<pre>ciocxemu.pl my.pkg.util</pre>	
Simple substitution on referencing package		<pre>ciocxemu.pl my.pkg.cx</pre>
After basic substitution, the packages are changed, but not the references between them	<pre>package my.pkg.util .EMU; import oracle.apps.cz. CIOEMU.IRuntimeNode; public class Toolkit { void formatDate() }</pre>	<pre>package my.pkg.cx.EMU; import oracle.apps.cz. CIOEMU.IRuntimeNode; import my.pkg.util.Toolkit; d = my.pkg.util.Toolkit.formatDate();</pre>
At runtime, the class reference is incorrect	<p>The new class is in a new package that uses the CIO Emulation API:</p> <pre>my.pkg.util.EMU .Toolkit</pre>	<p>The method call in the new class references the original package, which does not use the CIO Emulation API:</p> <pre>my.pkg.util.Toolkit</pre>

Cross-Package References with Custom Substitution

Stage	Class in Referenced Package	Class in Referencing Package
	<pre>my.pkg.util.Toolkit</pre>	<pre>my.pkg.cx.MyCXClass</pre>

Stage	Class in Referenced Package	Class in Referencing Package
Before simple substitution	<pre>package my.pkg.util; import oracle.apps.cz.cio.IRuntimeNode; public class Toolkit { void formatDate() }</pre>	<pre>package my.pkg.cx; import oracle.apps.cz.cio.IRuntimeNode; import my.pkg.util.Toolkit; d = my.pkg.util.Toolkit.formatDate();</pre>
Simple substitution on referenced package	<pre>ciocxemu.pl my.pkg.util</pre>	
Custom substitution on referencing package		<pre>ciocxemu.pl my.pkg.cx my.pkg.util my.pkg. util.emu</pre>
After custom substitution, the packages are changed, and also the references between them	<pre>package my.pkg.util .emu; import oracle.apps.cz. cioemu.IRuntimeNode; public class Toolkit { void formatDate() }</pre>	<pre>package my.pkg.cx.emu; import oracle.apps.cz. cioemu.IRuntimeNode; import my.pkg.util.emu .Toolkit; d = my.pkg.util.emu .Toolkit.formatDate();</pre>
At runtime, the class reference is correct	<p>The new class is in a new package that uses the CIO Emulation API:</p> <pre>my.pkg.util.emu .Toolkit</pre>	<p>The method call in the new class references the new package, which uses the CIO Emulation API:</p> <pre>my.pkg.util.emu .Toolkit</pre>

Important: Custom substitution does not perform all the modifications required to make your files ready for CIO Emulation for the FCE. It is possible that your packages will compile successfully but still contain incorrect references.

Output of the Script

When the substitution script runs, it creates the following output:

- A new subdirectory of the current working directory, named `emu`.
- In the `emu` subdirectory, the new Java files produced by the script, which have been modified as described in What the Substitution Script Does, page 6-11.
 - Your original Java files in the current working directory are left unchanged.
 - Files in the `emu` subdirectory are not overwritten. This behavior allows you to run the script multiple times over the same working directory without reconverting files that have already been converted. This avoids extra processing if you add files to the working directory after the initial run. It also preserves changes that you may have made to the `emu` files if you inadvertently run the script again on the working directory. See Errors from the Script, page 6-22 for more information.
- In the `emu` subdirectory, a log file that contains the informational messages that were written to the console by the script. The name of the log file is `CxEmuConversionyyyyymmdd.log`, where `yyyyymmdd` is the year, month, and date when the script was run.
- The header of the log file is shown in the example Conversion Log File: Header, page 6-21. It shows the `original_package` parameter that you specified. It also shows your `custom_target` and `custom_replacement` parameters, if any, or NOT PROVIDED otherwise.

Example 5. Conversion Log File: Header

```
Conversion to Cio Cx Emulation: Started at    2009/03/06 10:46:23
Input  Directory: /my/pkg/cx/
Output Directory: /my/pkg/cx/emu/
Input  Package   : my.pkg.cx
Output Package   : my.pkg.cx.emu
Input  Custom    : my.pkg.util
Output Custom    : my.pkg.util.emu
```

- A typical log file entry for a file being converted is shown in the example Conversion Log File: File Entry, page 6-21. The warning message indicates that the original file contained a CIO class that is not fully supported in emulation. See Errors from the Script, page 6-22 for more information.

Example 6. Conversion Log File: File Entry

```
=====
==
Convert: /my/pkg/cx/AddClonedInstances.java to
/my/pkg/cx/emu/AddClonedInstances.java
WARNING: At line: 13. oracle.apps.cz.cioemu.TransactionException.
Not supported but provided for CX compilation. Review usage and
behavior.
Completed: /my/pkg/cx/AddClonedInstances.java
```

- The concluding block in the log file is shown in the example Conversion Log File: Tail, page 6-22. The Pre Directory Java File Count is the number of Java source files in the current working directory when you start the substitution script.

All of those files are considered for processing when the script runs. The Converted CX File Count is the number of files actually created by the script in the emu subdirectory. If Converted CX File Count is smaller than Pre Directory Java File Count, it should indicate that some files already have converted versions in the emu subdirectory, and thus were not overwritten.

Example 7. Conversion Log File: Tail

```
Pre Directory Java File Count: 28
Converted CX File Count      : 28
Conversion to Cio Cx Emulation: Completed at 2009/03/06 10:46:44
```

Errors from the Script

When the substitution runs, it may generate some of the messages described in the table Substitution Script Error Messages, page 6-22.

Substitution Script Error Messages

Message	Explanation
Custom substitution: New text is required	If you provided more than one parameter, the script assumes that you are using custom substitution, which requires three arguments. See Custom Substitution, page 6-18.
WARNING: <i>path/emu/FileName.java</i> already exists. File not converted	A file with the same name as one being converted already exists in the emu subdirectory. Such files are not overwritten.
The import was converted. References to unsupported classes will not compile.	If your converted code refers to classes that do not have equivalents in the CIO Emulation API, it will not compile. See Differences Between FCE and CIO Emulation, page 6-4.
WARNING At line: <i>n</i> . <i>oracle.apps.cz.cioemu.ClassName</i> . Not supported but provided for CX compilation. Review usage and behavior	Your original file refers to a CIO class that is not fully supported in emulation.

The above table omits self-explanatory error messages such as "Incorrect number of arguments".

Compiling and Archiving Converted Files

After you run the substitution script to modify your Java class source files, compile them and archive them so that the classes can be used by Configurator Extension Rules in your configuration model.

Procedure

To compile and archive your converted files:

1. Compile the converted Java source files created by the substitution script in the `emu` subdirectory.

It is likely that some of your class references could not be resolved by the substitution script. See *About CIO Emulation for the FCE*, page 6-1 for an understanding of what the CIO Emulation API can provide.

2. Resolve any compilation errors.
3. Create a Java archive file for the compiled classes, in the usual way.

See the *Oracle Configurator Extensions and Interface Object Developer's Guide* for information on developing Java classes and archives.

4. Proceed to the next step under Tasks for Implementing CIO Emulation for the FCE, page 6-10.

Converting Configurator Extension Rules

After the substitution script has modified your Java class source files to refer to the CIO Emulation API, it is still necessary to modify the bindings in all of the Configurator Extension (CX) Rules that used your original classes, so that they, too, refer to the CIO Emulation API. This conversion of Rules is performed in Oracle Configurator Developer, as described in this section.

Assumptions

This task assumes the following:

- You have converted your Models to use the FCE, using the Model Conversion Utility.
- You have run the substitution script on your Java source files for your Configurator Extensions, and have compiled and archived your resulting Java class files (which now refer to the CIO Emulation API).
- Your converted Models contain CX Rules (which at this point still refer to the CIO API rather than the CIO Emulation API or Core API).

Limitations

Take note of the following limitations of the Rule conversion operation:

- You can only convert the CX Rules for one Model at a time, and this Model must already have been converted to use the FCE.
- All of the CX Rules in your Model are processed for conversion. You cannot include

or exclude specified CX Rules from conversion, since CIO-based CX Rules that are not converted cannot operate with your converted FCE Model and are thus no longer useful.

- CX Rules marked as Disabled are converted, and remain Disabled after conversion. Disabling a CX Rule does not exclude it from conversion.
- CX Rules are converted without creating backup copies in your converted FCE Model. The original CX Rules remain in your original unconverted Model, but are not retained in your converted FCE Model.

Procedure

To convert your CX Rules:

1. Log in to Oracle Configurator Developer.
2. Create a Configurator Extension (CX) Archive, using the Java class archive that you created for your modified source files.

See the *Oracle Configurator Developer User's Guide* for information on creating CX Archives.

Tip: As you proceed with converting your CX Rules, you can use this new CX Archive for all the Models whose CX Rules referenced your original classes.

3. Edit a Model that contains CX Rules that may need to be converted, and go to the General area of the Workbench.

If a Model has CX Rules that need conversion, then the General area of the Workbench includes a new section named Configurator Extensions Status. The section displays a message stating that the Configurator Extension Rules in the model were created with bindings whose arguments refer to Original Configurator Engine classes (that is, to the CIO API). Such Rules need to be converted to use the CIO Emulation API that are referenced in the new CX Archive.

4. If the Configurator Extensions Status section indicates a need for conversion, click **Convert**.

You need to add your new Emulation-based CX Archive to the Archive Path for your Model. The Edit Archive Path page appears, listing all CX Archives in the Main area of the Repository, subject to the current focus level and folder expansion.

5. Locate your new Emulation-based CX Archive and select it.

Click the check box in the Select column to select an Archive.

6. Click **Add to Selected List**.

The selected Archive appears at the end of the Selected List, meaning that it has been placed at the end of the Archive Path. See the section of the *Oracle Configurator Developer User's Guide* about archive path precedence for important information about the Archive Path.

7. When you are satisfied with the order of Archives in the Selected List, click **Apply**. This step triggers the actual conversion of the Rules.

When the conversion operation finishes, you return to the General area of the Workbench.

If the Rule conversion was not successful, then the Configurator Extensions Status section still appears, and an error message appears in the information area.

If the Rule conversion was successful, then the Configurator Extensions Status section no longer appears, and a message in the information area tells you that the Configurator Extension bindings in this model were successfully converted to use the classes for the Fusion Configurator Engine.

Important: The CX Rule conversion operation ensures that your Rules are modified to use the appropriate classes and interfaces from the CIO Emulation package. However, you must review your converted rules to verify their expected behavior.

8. If you examine the definitions of your CX Rules, you will find that they have been modified in accordance with the changes shown in the table Substitutions Performed by the Script, page 6-11.
 - The package name for the **Java Class** has been changed to refer to the package created by the substitution script. For example, `my.pkg.cx.MyCXClass` is changed to `my.pkg.cx.emu.MyCXClass`.
 - In the event bindings, any references to CIO API objects have been changed to refer to CIO Emulation API objects. For example, `oracle.apps.cz.cio.IRuntimeNode` is changed to `oracle.apps.cz.cioemu.IRuntimeNode`.

9. Repeat steps 3 through 7 for your other converted FCE Models.

Tip: If the set of classes in your new Emulation-based CX Archive corresponds to the set of classes in your old CIO-based CX Archive, then you can more easily determine which Models used the CIO classes (and thus need to have their CX Rules converted to use the CIO Emulation API classes referenced in the new Archive) by examining the list of Models Referencing This Archive in the old Archive.

10. Proceed to the next step under Tasks for Implementing CIO Emulation for the FCE, page 6-10.

Verifying Post-Conversion Behavior

To ensure the success of your conversion to CIO Emulation for the FCE, you must verify that your Configurator Extensions behave as expected after you have converted their code to use the emulation classes. Some suggestions for performing this verification are:

- Use logging in your Configurator Extension code, as described in the Oracle Configurator Extensions and Interface Object Developer's Guide. Logging output can help verify that a block of code was executed as expected. Your converted code should be using the method
`oracle.apps.cz.cioemu.Configuration.writeCXLogEntry()`.
- Ensure that all of the uses of Configurator Extensions in your application's UI are exercised.
- Review your converted Configurator Extension Rules in Oracle Configurator Developer.
- Review Differences Between FCE and CIO Emulation, page 6-4.

Glossary

This glossary contains definitions relevant to working with Oracle Configurator.

A

Archive Path

The ordered sequence of Configurator Extension Archives for a Model that determines which Java classes are loaded for Configurator Extensions and in what order.

B

base node

The node in a Model that is associated with a Configurator Extension Rule. Used to determine the event scope for a Configurator Extension.

batch validation

A background process for validating selections in a configuration.

binding

Part of a Configurator Extension Rule that associates a specified event with a chosen method of a Java class. *See also* event.

BOM item

The node imported into Oracle Configurator Developer that corresponds to an Oracle Bills of Material item. Can be a BOM Model, BOM Option Class node, or BOM Standard Item node.

BOM Model

A model that you import from Oracle Bills of Material into Oracle Configurator Developer. When you import a BOM Model, effective dates, ATO (Assemble To Order) rules, and other data are also imported into Configurator Developer. In Configurator Developer, you can extend the structure of the BOM Model, but you cannot modify the BOM Model itself or any of its attributes.

BOM Model node

The imported node in Oracle Configurator Developer that corresponds to a BOM Model created in Oracle Bills of Material.

BOM Option Class node

The imported node in Oracle Configurator Developer that corresponds to a BOM Option Class created in Oracle Bills of Material.

BOM Standard Item node

The imported node in Oracle Configurator Developer that corresponds to a BOM Standard Item created in Oracle Bills of Material.

Boolean Feature

An element of a component in the Model that has two options: true or false.

C**CDL (Constraint Definition Language)**

A language for entering configuration rules as text rather than assembling them interactively in Oracle Configurator Developer. CDL can express more complex constraining relationships than interactively defined configuration rules can.

The CIO is the API that supports creating and navigating the Model, querying and modifying selection states, and saving and restoring configurations.

CIO (Oracle Configuration Interface Object)

A server in the runtime application that creates and manages the interface between the client (usually a user interface) and the underlying representation of model structure and rules in the generated logic.

command event

An event that is defined by a character string and detected by a command listener.

Comparison Rule

An Oracle Configurator Developer rule type that establishes a relationship to determine the selection state of a logical Item (Option, Boolean Feature, or List-of-Options Feature) based on a comparison of two numeric values (numeric Features, Totals, Resources, Option counts, or numeric constants). The numeric values being compared can be computed or they can be discrete intervals in a continuous numeric input.

Compatibility Rule

An Oracle Configurator Developer rule type that establishes a relationship among Features in the Model to control the allowable combinations of Options. *See also,*

Property-based Compatibility Rule.

Compatibility Table

A kind of Explicit Compatibility Rule. For example, a type of compatibility relationship where the allowable combination of Options are explicitly enumerated.

component

A piece of something or a configurable element in a model such as a BOM Model, Model, or Component.

Component

An element of the model structure, typically containing Features, that is configurable and instantiable. An Oracle Configurator Developer node type that represents a configurable element of a Model.

Component Set

An element of the Model that contains a number of instantiated Components of the same type, where each Component of the set is independently configured.

configuration

A specific set of specifications for a product, resulting from selections made in a runtime configurator.

configuration attribute

A characteristic of an item that is defined in the host application (outside of its inventory of items), in the Model, or captured during a configuration session. Configuration attributes are inputs from or outputs to the host application at initialization and termination of the configuration session, respectively.

configuration model

Represents all possible configurations of the available options, and consists of model structure and rules. It also commonly includes User Interface definitions and Configurator Extensions. A configuration model is usually accessed in a runtime Oracle Configurator window. *See also* model.

configuration rule

A Logic Rule, Compatibility Rule, Comparison Rule, Numeric Rule, Design Chart, Statement Rule, or Configurator Extension rule available in Oracle Configurator Developer for defining configurations. *See also* rules.

configuration session

The time from launching or invoking to exiting Oracle Configurator, during which end users make selections to configure an orderable product. A configuration session is

limited to one configuration model that is loaded when the session is initialized.

configurator

The part of an application that provides custom configuration capabilities. Commonly, a window that can be launched from a host application so end users can make selections resulting in valid configurations. *Compare* Oracle Configurator.

Configurator Developer

See OCD.

Configurator Extension

An extension to the configuration model beyond what can be implemented in Configurator Developer.

A type of configuration rule that associates a node, Java class, and event binding so that the rule operates when an event occurs during a configuration session.

A Java class that provides methods that can be used to perform configuration actions.

Configurator Extension Archive

An object in the Repository that stores one or more compiled Java classes that implement Configurator Extensions.

connectivity

The connection across components of a model that allows modeling such products as networks and material processing systems.

Connector

The node in the model structure that enables an end user at runtime to connect the Connector node's parent to a referenced Model.

Constraint Definition Language

See CDL

Container Model

A type of BOM Model that you import from Oracle Bills of Material into Oracle Configurator Developer to create configuration models that support connectivity and contain trackable components. Configurations created from Container Models can be tracked and updated in Oracle Install Base

Contributes to

A relation used to create a specific type of Numeric Rule that accumulates a total value. *See also* Total.

Consumes from

A relation used to create a specific type of Numeric Rule that decrements a total value, such as specifying the quantity of a Resource used.

count

The number or quantity of something, such as selected options. *Compare* instance.

CZ

The product shortname for Oracle Configurator in Oracle Applications.

CZ schema

The implementation version of the standard runtime Oracle Configurator data-warehousing schema that manages data for the configuration model. The implementation schema includes all the data required for the runtime system, as well as specific tables used during the construction of the configurator.

D**default**

In a configuration, the automatic selection of an option based on the preselection rules or the selection of another option.

Defaults relation

An Oracle Configurator Developer Logic Rule relation that determines the logic state of Features or Options in a default relation to other Features and Options. For example, if A Defaults B, and you select A, B becomes Logic True (selected) if it is available (not Logic False).

Design Chart

An Oracle Configurator Developer rule type for defining advanced Explicit Compatibilities interactively in a table view.

E**element**

Any entity within a model, such as Options, Totals, Resources, UI controls, and components.

end user

The ultimate user of the runtime Oracle Configurator. The types of end users vary by project but may include salespeople or distributors, administrative office staff, marketing personnel, order entry personnel, product engineers, or customers directly

accessing the application via a Web browser or kiosk. *Compare* user.

event

An action or condition that occurs in a configuration session and can be detected by a listener. Example events are a change in the value of a node, the creation of a component instance, or the saving of a configuration. The part of model structure inside which a listener listens for an event is called the event binding scope. The part of model structure that is the source of an event is called the event execution scope. *See also* command event.

Excludes relation

An Oracle Configurator Developer Logic Rule type that determines the logic state of Features or Options in an excluding relation to other Features and Options. For example, if A Excludes B, and if you select A, B becomes Logic False, since it is not allowed when A is true (either User or Logic True). If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or Unknown. *See* Negates relation.

F

feature

A characteristic of something, or a configurable element of a component at runtime.

Feature

An element of the model structure. Features can either have a value (numeric or Boolean) or enumerated Options.

G

generated logic

The compiled structure and rules of a configuration model that is loaded into memory on the Web server at configuration session initialization and used by the Oracle Configurator engine to validate runtime selections. The logic must be generated either in Oracle Configurator Developer or programmatically in order to access the configuration model at runtime.

guided buying or selling

Needs assessment questions in the runtime UI to guide and facilitate the configuration process. Also, the model structure that defines these questions. Typically, guided selling questions trigger configuration rules that automatically select some product options and exclude others based on the end user's responses.

H

host application

An application within which Oracle Configurator is embedded as integrated functionality, such as Order Management or iStore.

I**implementer**

The person who uses Oracle Configurator Developer to build the model structure, rules, and UI customizations that make up a runtime Oracle Configurator. Commonly also responsible for enabling the integration of Oracle Configurator in a host application.

Implies relation

An Oracle Configurator Developer Logic Rule type that determines the logic state of Features or Options in an implied relation to other Features and Options. For example, if A Implies B, and you select A, B becomes Logic True. If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or Unknown. *See* Requires relation.

import server

A database instance that serves as a source of data for Oracle Configurator's Populate, Refresh, Migrate, and Synchronization concurrent processes. The import server is sometimes referred to as the remote server.

initialization message

The XML (Extensible Markup Language) message sent from a host application to the Oracle Configurator Servlet, containing data needed to initialize the runtime Oracle Configurator. *See also* termination message.

instance

A runtime occurrence of a component in a configuration that is determined by the component node's Instance attribute specifying a minimum and maximum value. *See also* instantiate. *Compare* count.

Also, the memory and processes of a database.

instantiate

To create an instance of something. Commonly, to create an instance of a component in the runtime user interface of a configuration model.

item

A product or part of a product that is in inventory and can be delivered to customers.

Item

A Model or part of a Model that is defined in the Item Master. Also data defined in Oracle Inventory.

Item Master

Data stored to structure the Model. Data in the CZ schema Item Master is either entered manually in Oracle Configurator Developer or imported from Oracle Applications or a legacy system.

Item Type

Data used to classify the Items in the Item Master. Item Catalogs imported from Oracle Inventory are Item Types in Oracle Configurator Developer.

L**listener**

A class in the CIO that detects the occurrence of specified events in a configuration session.

Logic Rule

An Oracle Configurator Developer rule type that expresses constraint among model elements in terms of logic relationships. Logic Rules directly or indirectly set the logical state (User or Logic True, User or Logic False, or Unknown) of Features and Options in the Model.

There are four primary Logic Rule relations: Implies, Requires, Excludes, and Negates. Each of these rules takes a list of Features or Options as operands. *See also* Implies relation, Requires relation, Excludes relation, and Negates relation.

M**model**

A generic term for data representing products. A model contains elements that correspond to items. Elements may be components of other objects used to define products. A configuration model is a specific kind of model whose elements can be configured by accessing an Oracle Configurator window.

Model

The entire hierarchical "tree" view of all the data required for configurations, including model structure, variables such as Resources and Totals, and elements in support of intermediary rules. Includes both imported BOM Models and Models created in Configurator Developer. May consist of BOM Option Classes and BOM Standard Items.

model structure

Hierarchical "tree" view of data composed of elements (Models, Components, Features, Options, BOM Models, BOM Option Class nodes, BOM Standard Item nodes, Resources, and Totals). May include reusable components (References).

N

Negates relation

A type of Oracle Configurator Developer Logic Rule type that determines the logic state of Features or Options in a negating relation to other Features and Options. For example, if one option in the relationship is selected, the other option must be Logic False (not selected). Similarly, if you deselect one option in the relationship, the other option must be Logic True (selected). *Compare* Excludes relation.

node

The icon or location in a Model tree in Oracle Configurator Developer that represents a Component, Feature, Option or variable (Total or Resource), Connector, Reference, BOM Model, BOM Option Class node, or BOM Standard Item.

Numeric Rule

An Oracle Configurator Developer rule type that expresses constraint among model elements in terms of numeric relationships. *See also*, Contributes to and Consumes from.

O

object

Entities in Oracle Configurator Developer, such as Models, Usages, Properties, Effectivity Sets, UI Templates, and so on. *See also* element.

OCD

See Oracle Configurator Developer.

option

A logical selection made in the Model Debugger or a runtime Oracle Configurator by the end user or a rule when configuring a component.

Option

An element of the Model. A choice for the value of an enumerated Feature.

Oracle Configurator

The product consisting of development tools and runtime applications such as the CZ schema, Oracle Configurator Developer, and runtime Oracle Configurator. Also the

runtime Oracle Configurator variously packaged for use in networked or Web deployments.

Oracle Configurator Developer

The tool in the Oracle Configurator product used for constructing and maintaining configuration models.

Oracle Configurator engine

The part of the Oracle Configurator product that uses configuration rules to validate runtime selections. Compare generated logic. *See also* generated logic.

Oracle Configurator schema

See CZ schema.

Oracle Configurator Servlet

A Java servlet that participates in rendering legacy user interfaces for Oracle Configurator.

Oracle Configurator window

The user interface that is launched by accessing a configuration model and used by end users to make the selections of a configuration.

P

Populator

An entity in Oracle Configurator Developer that creates Component, Feature, and Option nodes from information in the Item Master.

Property

A named value associated with a node in the Model or the Item Master. A set of Properties may be associated with an Item Type. After importing a BOM Model, Oracle Inventory Catalog Descriptive Elements are Properties in Oracle Configurator Developer.

Property-based Compatibility Rule

An Oracle Configurator Developer Compatibility Rule type that expresses a kind of compatibility relationship where the allowable combinations of Options are specified implicitly by relationships among Property values of the Options.

publication

A unique deployment of a configuration model (and optionally a user interface) that enables a developer to control its availability from host applications such as Oracle Order Management or iStore. Multiple publications can exist for the same configuration

model, but each publication corresponds to only one Model and User Interface.

publishing

The process of creating a publication record in Oracle Configurator Developer, which includes specifying applicability parameters to control runtime availability and running an Oracle Applications concurrent process to copy data to a specific database.

R

reference

The ability to reuse an existing Model or Component within the structure of another Model (for example, as a subassembly).

Reference

An Oracle Configurator Developer node type that denotes a reference to another Model.

Repository

Set of pages in Oracle Configurator Developer that contains areas for organizing and maintaining Models and shared objects in a single location.

Requires relation

An Oracle Configurator Developer Logic Rule relationship that determines the logic state of Features or Options in a requirement relation to other Features and Options. For example, if A Requires B, and if you select A, B is set to Logic True (selected). Similarly, if you deselect A, B is set to Logic False (deselected). *See* Implies relation.

Resource

A variable in the Model used to keep track of a quantity or supply, such as the amount of memory in a computer. The value of a Resource can be positive or zero, and can have an Initial Value setting. An error message appears at runtime when the value of a Resource becomes negative, which indicates it has been over-consumed. Use Numeric Rules to contribute to and consume from a Resource.

Also a specific node type in Oracle Configurator Developer. *See also* node.

rules

Also called business rules or configuration rules. In the context of Oracle Configurator and CDL, a rule is not a business rule. Constraints applied among elements of the product to ensure that defined relationships are preserved during configuration. Elements of the product are Components, Features, and Options. Rules express logic, numeric parameters, implicit compatibility, or explicit compatibility. Rules provide preselection and validation capability in Oracle Configurator.

See also Comparison Rule, Compatibility Rule, Design Chart, Logic Rule and Numeric Rule.

runtime

The environment in which an implementer (tester), end user, or customer configures a product whose model was developed in Oracle Configurator Developer. *See also* configuration session.

S**Statement Rule**

An Oracle Configurator Developer rule type defined by using the Oracle Configurator Constraint Definition Language (text) rather than interactively assembling the rule's elements.

T**termination message**

The XML (Extensible Markup Language) message sent from the Oracle Configurator Servlet to a host application after a configuration session, containing configuration outputs. *See also* initialization message.

Total

A variable in the Model used to accumulate a numeric total, such as total price or total weight.

Also a specific node type in Oracle Configurator Developer. *See also* node.

U**UI**

See User Interface.

UI Templates

Templates available in Oracle Configurator Developer for specifying UI definitions.

Unknown

The logic state that is neither true nor false, but unknown at the time a configuration session begins or when a Logic Rule is executed. This logic state is also referred to as Available, especially when considered from the point of view of the runtime Oracle Configurator end user.

user

The person using a product or system. Used to describe the person using Oracle Configurator Developer tools and methods to build a runtime Oracle Configurator. *Compare* end user.

user interface

The visible part of the application, including menus, dialog boxes, and other on-screen elements. The part of a system where the user interacts with the software. Not necessarily generated in Oracle Configurator Developer. *See also* User Interface.

User Interface

The part of an Oracle Configurator implementation that provides the graphical views necessary to create configurations interactively. A user interface is generated from the model structure. It interacts with the model definition and the generated logic to give end users access to customer requirements gathering, product selection, and any extensions that may have been implemented. *See also* UI Templates.

V**validation**

Tests that ensure that configured components will meet specific criteria set by an enterprise, such as that the components can be ordered or manufactured.

W**Workbench**

Set of pages in Oracle Configurator Developer for creating, editing, and working with Repository objects such as Models and UI Templates.

Index

A

Accumulator Rules

- definition, 3-32

actions

- Auto-Complete Configuration, 3-62
- Cancel Processing, 3-62
- Cancel Request, 3-63
- Configure Instance, 3-63
- Copy Instance, 3-63
- Create and Go to Instance, 3-63
- Create Instance, 3-63
- Delete Instance, 3-63
- Override Conflict, 3-63
- processing page, 3-64
- Remove Instance, 3-63
- Save and Exit, 3-64
- Undo Auto-Complete, 3-64

ATPChangedByAC System Property
, 3-15

Auto-Complete Configuration

- action, 3-62
- definition, 1-3
- runtime behavior, 4-2

AutoCompleteSuccessful System Property, 3-13

AutoOverrideEnabled System Property, 3-15

B

bidirectional (rule)

- definition, 3-30

binding

- variable, 1-4

BOM Nodes

- relative quantity, 3-22
- unique characteristics in FCE Models, 3-22

C

Cancel Processing

- action, 3-62

Cancel Request

- action, 3-63

cardinality

- instances, 4-5

ChangedByAC System Property, 3-11

CLASS_SEQ, 3-45

Compatibility Rules, 3-32

CONFIG_ENGINE_TYPE, 3-46

ConfigComplete System Property, 3-12

ConfigurationChangedByAC System Property, 3-14

Configurator Engine setting

- description, 3-2

Configurator Extensions, 3-34

Configure Instance

- action, 3-63

ConflictOverridable System Property, 3-16

ConnectionCount

- System Property, 3-6

Connectors

- availability, 1-2, 2-9, 3-24
- description, 3-24
- reverse connectors, 3-24

constraints

- definition, 1-1

- soft constraints, 1-2
- Copy Instance
 - action, 3-63
- Create and Go to Instance
 - action, 3-63
- Create Instance
 - action, 3-63

D

- Defaults
 - Rule Class, 3-30
- DefinitionMaxConnections
 - System Property, 3-8
- DefinitionMaxInstances System Property, 3-8
- DefinitionMaxQuantity System Property, 3-6
- DefinitionMaxRelQuantity System Property, 3-7
- DefinitionMaxSelected System Property, 3-7
- DefinitionMaxValue System Property, 3-6
- DefinitionMinConnections
 - System Property, 3-8
- DefinitionMinInstances System Property, 3-7
- DefinitionMinQuantity System Property, 3-6
- DefinitionMinRelQuantity System Property, 3-6
- DefinitionMinSelected System Property, 3-7
- DefinitionMinValue System Property, 3-6
- Delete Instance
 - action, 3-63
- documentation
 - related documents, xi
- domain
 - definition, 3-16
- Domain Ordering
 - setting, 3-16

E

- Effectivity
 - description, 3-26
- events
 - description
 - postAutoComplete, 3-36
 - preAutoComplete, 3-35

F

- Finish
 - runtime UI flow, 4-9

- Fundamental Conflict Message
 - description, 3-55
- Fusion Configurator Engine
 - features, 1-5

G

- General area of the Workbench, 3-2
- generic instance, 4-5

H

- HasItemsToAddress System Property, 3-12

I

- identifiable instances, 4-5
- images and icons
 - list and descriptions, 3-66
- importing
 - legacy rules , 3-45
 - statement rules , 3-45
- InAdjustMode System Property, 3-14
- InErrorMode System Property, 3-12
- Initial Values
 - description, 3-21
- InputRequired
 - System Property, 3-10
- InputRequiredError System Property, 3-11
- InputRequiredFlag
 - System Property, 3-10
- InputRequiredInSubtree System Property, 3-11
- InputRequired System Property, 3-12
- InstanceCount System Property, 1-5
- instance management, 1-3, 4-4
 - cardinality, 4-5
 - enhancements, 1-3, 4-4
 - generic instance, 4-5
 - identifiable instance, 4-5
 - placeholders, 4-5
- instance pool
 - definition, 4-7
- instances, 1-3, 4-4
- IsBound
 - System Property, 3-9
- IsBoundQuantity
 - System Property, 3-9
- IsBoundRelQuantity

- System Property, 3-10
- IsBoundSelectionMode
 - System Property, 3-9
- IsPlaceholder System Property, 3-61

L

- ListPriceChangedByAC System Property, 3-14
- Logic States, 3-27

M

- MaxConnections
 - System Property, 3-6
- MaxRelQuantity System Property, 3-5
- MinConnections
 - System Property, 3-5
- MinRelQuantity System Property, 3-5
- Model Node System Properties
 - use in FCE Models, 3-3
- Models
 - References, 3-3
- Model Structure
 - settings and attributes, 3-1

O

- overconstrained
 - Fundamental Conflict Message, 3-55
- Override Conflict
 - action, 3-63
- OverrideSuccessful System Property, 3-16

P

- placeholder, 4-5
- postAutoComplete (event), 3-36
- preAutoComplete (event), 3-35
- processing page, 3-64
- Profile Options
 - list, definitions, and default values, 2-1
- Properties
 - introduction, 3-3
- Proposed
 - System Property, 3-8

R

- References, 3-3

- relative quantity
 - definition, 3-22
 - displaying BOM relative quantities at runtime, 3-51
- RelativeQuantity System Property, 3-5
- Remove Instance
 - action, 3-63
- Require End-User Input
 - setting definition, 3-19
- Resources
 - description, 3-21
- restore
 - restoring a configuration, 4-11
- RULE_CLASS, 3-46
- Rule Classes
 - Defaults, 3-30
 - definition, 3-27
- rule import, 3-45
- rules
 - Accumulator Rules, 3-32
 - overview, 3-27
- Runtime
 - Auto-Complete Configuration, 4-2
 - flows and behavior, 4-2

S

- Save and Exit
 - action, 3-64
- SelectedCount System Property, 3-5
- SellingPriceChangedByAC System Property, 3-15
- soft constraints, 1-2
- statement rules
 - importing, 3-45
- System Properties
 - ATPChangedByAC, 3-15
 - AutoCompleteSuccessful, 3-13
 - AutoOverrideEnabled, 3-15
 - ChangedByAC, 3-11
 - ConfigComplete, 3-12
 - ConfigurationChangedByAC, 3-14
 - ConflictOverridable, 3-16
 - ConnectionCount, 3-6
 - DefinitionMaxConnections, 3-8
 - DefinitionMaxInstances, 3-8
 - DefinitionMaxQuantity, 3-6

DefinitionMaxRelQuantity, 3-7
DefinitionMaxSelected, 3-7
DefinitionMaxValue, 3-6
DefinitionMinConnections, 3-8
DefinitionMinInstances, 3-7
DefinitionMinQuantity, 3-6
DefinitionMinRelQuantity, 3-6
DefinitionMinSelected, 3-7
DefinitionMinValue, 3-6
HasItemsToAddress, 3-12
InAdjustMode, 3-14
InErrorMode, 3-12
InputRequired, 3-10, 3-12
InputRequiredError, 3-11
InputRequiredFlag, 3-10
InputRequiredInSubtree, 3-11
InstanceCount, 1-5
IsBound, 3-9
IsBoundQuantity, 3-9
IsBoundRelQuantity, 3-10
IsBoundSelectionState, 3-9
IsPlaceholder, 3-61
ListPriceChangedByAC, 3-14
MaxConnections, 3-6
MaxRelQuantity, 3-5
MinConnections, 3-5
MinRelQuantity, 3-5
mutable, 3-3
OverrideSuccessful, 3-16
Proposed, 3-8
RelativeQuantity, 3-5
SelectedCount, 3-5
SellingPriceChangedByAC, 3-15
ValidationErrorText, 3-11

T

Text Features
 definition, 3-20
Totals
 description, 3-21

U

unbound (variable), 1-4
 logic states, 3-27
Undo Auto-Complete
 action, 3-64

Undo Status Message Box
 UI Content Template, 3-56
User Decision
 definition, 3-30
user input required
 Require End-User Input setting, 3-19
User Interface actions
 list and definitions, 3-62
User Interface Content templates
 Connection Management Table, 3-53
User Interface Content Templates, 3-49
 Auto-Complete Status Button Bar, 3-57
 Auto-Complete Status Dialog, 3-52
 Basic Conflict Button Bar, 3-58
 BOM Item Status Region with Range Display, 3-52
 Button Bar Templates, 3-56
 Confirm Override Button Bar, 3-58
 Confirm Override Dialog Page, 3-55
 Decimal Input with Range Display, 3-50
 Fundamental Conflict Message, 3-55
 Generic Processing Page, 3-59
 Generic Processing Page with Stop Button, 3-60
 Input Required Button Bar, 3-56
 Input Required Dialog Page, 3-54
 Input Required Message Box, 3-54
 Instance Chooser Button Bar, 3-58
 Instance Chooser Page, 3-53
 Integer Input with Range Display, 3-50
 Item Selection Tables with Quantity and Range Display, 3-51
 Processing Page Button Bar, 3-57
 Undo Status Message Box, 3-56
 User Request Conflict Dialog Page, 3-55
User Interface Definition
 description, 3-67
User Interface elements
 Instance Chooser Table, 3-61
User Interface Master Templates, 3-48

V

ValidationErrorText System Property, 3-11
variable
 definition, 1-4