Oracle® Configurator Developer

User's Guide

Release 11*i*

Part No. B10614-01

Februrary 2003

This document describes how to build and deploy configuration models using Oracle Configurator Developer.

ORACLE®

Oracle Configurator Developer User's Guide, Release 11*i*

Part No. B10614-01

Primary Author: Stephen R. Damiani

Contributing Author: Tina Brand

Contributor: Mark Sawtelle, Harriet Shanzer, Oracle Configurator Development Group

Contents

2 Managing Models

3 About Models

4 Constructing Model Structure

5 Constructing Configuration Rules

6 Constructing the User Interface

7 Testing and Debugging

8 Using Repository Window Tools

9 Using Model Window Tools

List of Tables

List of Figures

List of Examples

Send Us Your Comments

Oracle Configurator Developer User's Guide, Release 11*i*

Part No. B10614-01

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

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

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

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

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

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

Preface

Welcome to the *Oracle Configurator Developer User's Guide*. This user's guide includes the information you need to work with Oracle Configurator Developer effectively.

Intended Audience

This guide assumes you have a working knowledge of your business processes, tools, and your product configurations. It also assumes you are familiar with configurator applications. If you have never used a configurator application, we suggest you attend one or more of the Oracle Configurator training classes available through Oracle University. You should also be familiar with Oracle Applications and the Oracle Applications database.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an

otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

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

Structure

This documentation contains:

- Chapter 1, "Introduction" includes information on planning your project to ensure success using Oracle Configurator Developer.
- Chapter 2, "Managing Models" explains how to create and organize models, Effectivity Sets, and Usages and publish models.
- Chapter 3, "About Models" describes the data and tools available to you when building a model in Configurator Developer and the role of a Model's structure when you ultimately implement your configuration model.
- Chapter 4, "Constructing Model Structure" describes how to use Oracle Configurator Developer to create the fundamental structure upon which your configuration models are built.
- Chapter 5, "Constructing Configuration Rules" provides information on defining rules that determine what options an end user can select to create a valid configuration.
- Chapter 6, "Constructing the User Interface" describes how to create a User Interface in Oracle Configurator Developer and use it to test configuration models.
- Chapter 7, "Testing and Debugging" includes information about the Test / Debug module which you use to test the product structure, rules, and User Interface for your configuration models.
- Chapter 8, "Using Repository Window Tools" explains the tools you use to manage configuration models and their shared attributes within Oracle Configurator Developer.
- Chapter 9, "Using Model Window Tools" describes the tools you use to create and update a model's product structure, configuration rules, and User Interface.

- Appendix A, "The Runtime Oracle Configurator" describes tasks commonly performed when configuring an item in a runtime Oracle Configurator.
- "Glossary of Terms and Acronyms" contains definitions that you may need while working with Oracle Configurator.

Related Documents

For more information, see the following manuals in Release 11*i* of the Oracle Product documentation set:

- *Oracle Configurator Release Notes*
- *Oracle Configurator Installation Guide*
- *Oracle Configurator Implementation Guide*
- *Oracle Configuration Interface Object (CIO) Developer's Guide*
- *Oracle Bills of Material User's Guide*
- *Oracle Inventory User's Guide*
- *Oracle Configurator Methodologies*
- Oracle Applications Release 11*i* documentation

Conventions

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

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

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a new term, a term defined in the glossary, specific keys, and labels of user interface objects. Boldface type also indicates a menu, command, or option, especially within procedures

Convention	Meaning
italics	Italic type in text, tables, or code examples indicates user-supplied text. Replace these placeholders with a specific value or string.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
>	The left bracket alone represents the MS DOS prompt.
$	The dollar sign represents the DIGITAL Command Language prompt in Windows and the Bourne shell prompt in Digital UNIX.
%	The per cent sign alone represents the UNIX prompt.
`name()`	In text other than code examples, the names of programming language methods and functions are shown with trailing parentheses. The parentheses are always shown as empty. For the actual argument or parameter list, see the reference documentation. This convention is *not* used in code examples.

Getting Help with Oracle Configurator Developer

Help for Oracle Configurator Developer is available from the Help menu online from the menu bar and has the same content as the *Oracle Configurator Developer User's Guide*.

Product Support

The mission of the Oracle Support Services organization is to help you resolve any issues or questions that you have regarding Oracle Configurator Developer and Oracle Configurator.

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

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

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

1. Choose the **TARs** link in the left menu.
2. Click on **Create a TAR**.
3. Fill in or choose a profile.
4. In the same form:

a. Choose **Product**: Oracle Configurator or Oracle Configurator Developer

b. Choose **Type of Problem**: Oracle Configurator Generic Issue template

5. Provide the information requested in the iTAR template.

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

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

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

1

Introduction

Oracle Configurator Developer is the development tool in the Oracle Configurator family of products. It provides a convenient drag-and-drop interface that enables you to rapidly develop a configurator. A **configurator** is a tool for configuring products and services. The configuration process can include assessing customer needs, selecting product and service components, and viewing configurations.

A configurator enables end users to access the parts that make up your product and the rules that govern how those parts fit together. With a configurator, end users can generate any custom product configuration that the rules allow. A configurator brings the expertise of your enterprise to the point of sale, dramatically changing and improving the way you sell products and services.

1.1 The Runtime Oracle Configurator

With Oracle Configurator Developer, you build a Model, configuration rules, and User Interface structure that reflect your enterprise and your end users' requirements. The Model, all configuration rules, and User Interface structure are stored in the CZ schema in the Oracle Applications database. The CZ schema is part of the Oracle Applications database. There is generally one CZ schema per Oracle Applications database instance.

The compiled configuration rules and Model structure exist as the **Active Model** in the CZ schema. The Active Model enforces valid configurations based on end user selections. The User Interface definitions of the configuration model function as the **User Interface**. The User Interface also interprets the data in the CZ schema and keeps the UI state current as the end user works. In other words, when the end user works in the Oracle runtime Configurator, the CZ schema, the Active Model, and the User Interface determine what is available for selection, what results from selections, and how it is displayed.

Configurator Developer provides a generic structure and a look and feel for the Oracle runtime Configurator that is enforced by the Active Model, UI, and CZ schema. This generic User Interface can also be customized (see Section 6.3, "Customizing the Default User Interface" on page 6-20).

The runtime Configurator you create with Oracle Configurator Developer can be deployed as:

- A DHTML Oracle Configurator window in Oracle Order Management (OM), *i*Store, Sales Online, Order Capture, or a custom Web application
- DHTML or a Java applet Oracle Configurator window in Order Management, TeleSales, or Oracle Order Capture

Further information on these deployment options is available in the Oracle Applications Help system. See the *Oracle Configurator Implementation Guide* for more information on the mechanics of deployment.

Oracle Configurator is integrated with Oracle Applications so that an end user can configure a product based on an Oracle Bill of Materials (BOM). Oracle Configurator dynamically creates a configuration model that reflects BOM rules, including parent-child, optional or required selections, mutually exclusive selections, and Quantity Cascade rules. If you want to add additional Model structure to the BOM Model, add additional rules, or customize the User Interface, you must do so in Configurator Developer.

1.2 Plan your Project

You need to plan carefully before you begin to build a configurator in Oracle Configurator Developer. Here are some points you should consider during planning. For additional warnings and helpful hints, see the *Oracle Configurator Release Notes*.

- Plan on attending a training course in using Configurator Developer.
- Plan to meet all platform requirements for Configurator Developer and your Oracle runtime Configurator as presented in the *Oracle Configurator Installation Guide*.
- Plan on setting up your Configurator Developer users on a local area network (LAN). Do not attempt to connect Configurator Developer clients to a database running on a remote server over a wide area network (WAN), T1 connection, or modem due to performance issues. A WAN configuration may be practical if the network bandwidth is high enough. However, in many situations, it will be

necessary to run Configurator Developer remotely using a Windows Terminal Server collocated with the database server.

- Plan to express your requirements for valid configurations in terms of the rules that Configurator Developer provides. See Section 1.2.2, "Design your Configuration Rules" on page 1-4.
- Plan your User Interface. If you plan to deploy your configuration model as a DHTML window, design a template to contain the Configurator frameset at runtime, the colors, banners, and buttons. Sample HTML files for a template that you can customize are installed as part of your Oracle Configurator installation (see the *Oracle Configurator Implementation Guide*).
- For custom deployments that are not integrated with Oracle Applications, gather the requirements for needed outputs such as quotes, proposals, and order entry data, their format, and the data for populating them. Oracle Configurator provides pre-defined output for Oracle ERP orders.
- For custom deployments that are not integrated in Oracle Applications, gather the requirements for integrating your system with other systems such as data synchronization and replication, quotes, and orders.
- Determine your requirements for deployment beyond those stated in Oracle Configurator documentation.
- Plan to define your product Model first, before defining configuration rules.
- Establish standardized and meaningful naming conventions for Model nodes and Rules. Use meaningful names for Components and Options. Names like **Response 1** and **Response 2** can easily lead to confusion among developers as well as end users.

You also need to do the basic design work for your configuration model. You need to consider what functionality your end users require. For example, you may need to add a customer needs assessment component. For more information, see Chapter 6, "Constructing the User Interface".

1.2.1 Identify your Product Data

Oracle Configurator Developer requires that an CZ schema be identified as a data source. You must populate the CZ schema **Item Master** with enterprise data needed for product configuration. To this end, you need to identify the source of your product data and structure. If your data comes from Oracle BOM, identify which bills to import into Configurator. You import data from Oracle BOM using concurrent requests. See the *Oracle Configurator Implementation Guide*.

If your data comes from Oracle Inventory Item data, or from an external data source, you must develop a mechanism for populating the Configurator import tables, and a plan for refreshing the import as required. Your Database Administrator (DBA) may prepare existing enterprise data for import.

If property data is imported from Oracle as part of the standard BOM import, properties and their values must be represented as Catalog Descriptive Elements and Descriptive Element Values in Oracle Inventory. (See the *Oracle Inventory User's Guide* for more information about Descriptive Elements.)

For data that is not imported, the configurator project team manually populates the CZ schema Item Master within Configurator Developer.

For more information about preparing and importing data, see the *Oracle Configurator Implementation Guide* or your DBA. For information on setting up a data source for Configurator Developer to connect to, see the *Oracle Configurator Installation Guide*.

1.2.2 Design your Configuration Rules

You must consider what rules you need to build into your configuration model. The design step may include writing a functional specification and other design documents.

Ask yourself questions such as:

- What components must be included in a valid configuration?
- What components are optional?
- What sub-components are compatible with each other?
- What selections affect another selection?
- What are valid default initial selections?
- What rules define the configuration of product families?
- What rules define the relations among product families?

1.2.3 Multi-User Development Strategy

If you need to have more than one developer working on a configuration Model, consider the following suggestions.

- To construct your configuration model efficiently, ensure that only one Configurator Developer user at a time makes changes to the Item Master that

you import into the CZ schema. Changes to the Item Master are written directly to the database, so if more than one person is working on the Item Master at the same time, the last update overwrites any previous work.

- When the Model structure is complete, coordinate the rule definition activities so different developers are working on separate, non-overlapping parts of a total set of rules.
- Oracle Configurator Developer's Model referencing feature enables you to use existing Models as subassemblies within a Model. Therefore, if a root-level Model has references to two other Models, two different developers can modify the referenced Models simultaneously and not interfere with anyone else's work.

1.3 Oracle Configurator Developer

Oracle Configurator Developer is made up of two major components: the Repository window and the Model window.

1.3.1 Repository Window

The **Repository** window provides a workspace for organizing Models and managing Model attributes such as Effectivity Sets and Usages. The Repository is an interface to your configuration Models that provides tools for managing Models and their shared attributes. You can also use this window to create **Model publications**, modify existing publications, and open a Model in the Model window for editing. The Model window provides tools for creating Model structure, configuration rules, and UI definition.

1.3.2 Model Window

You use the **Model** window to build a product Model and configuration rules that reflect the requirements of your product and business. Configurator Developer can automatically generate a User Interface based on the product Model. You can customize this interface to have the right look and feel for your enterprise.

The fundamental elements of a configurator built with Oracle Configurator Developer are:

- A product Model that captures the structure of your product, such as an imported BOM

- Configuration rules that constrain the relationships between parts of your product
- A User Interface that reflects the Model structure and defines the appearance of the runtime Oracle Configurator

1.4 The Overall Process

Implementing and maintaining an Oracle configurator consists of the following steps:

1. Setting up Oracle Configurator Developer
2. Importing data
3. Building and managing a configuration Model
4. Publishing and deploying the Model
5. Updating the Model when a new version of your product is available or you upgrade to a new release of Configurator Developer

1.4.1 Setup for Implementation

Oracle Configurator Developer is not currently part of Oracle Applications and therefore must be installed separately from the Oracle Configurator Developer compact disc. See the *Oracle Configurator Installation Guide* for details.

When you run Oracle Configurator Developer, you supply a username, password, and datasource. You can also run Oracle Configurator Developer using predefined parameters. This method allows you to provide preset values for the mandatory login parameters and bypass the Configurator Developer login screen. See the *Oracle Configurator Installation Guide* for details.

Your work in Oracle Configurator Developer is stored in the CZ schema, which is a schema of tables in the Oracle Applications database that are named with the CZ_ prefix. All changes you make are stored directly in the CZ schema as soon as you enter the information and press **Enter**, or click in another location in the User Interface. Therefore, you do not need to explicitly save your work. Additionally, note that there is no "undo" feature in Oracle Configurator Developer, so if, for example, you create a component and then decide you do not need it, you must manually delete it from the system.

For information on how to ensure that the CZ schema contains the data you need, see the *Oracle Configurator Implementation Guide* or your DBA.

Note: If you maintain both a development and a production database, do not run Configurator Developer on the Oracle Configurator schema in your production instance. For more information, see the *Oracle Configurator Implementation Guide*.

1.4.2 Managing Models

A configuration model precisely describes a logical configuration of a product that is added to a sales order and ultimately shipped to a customer. You create Models in Configurator Developer based on products that can be configured according to validation rules that you define. When purchasing a product, customer requirements are collected by a configurator, which uses the rules you defined and your Model definition to generate a configuration. Managing Models in Configurator Developer includes organizing them into Folders, creating Model publications to make them available to a hosting application, and creating and maintaining Usages and Effectivity Sets.

A Model can also consist of submodels. A single submodel can be included in more than one Model by creating a **Reference**. A single Model can include definitions for multiple user interfaces serving various types of end users. One Model can be assigned to multiple Usages and Effectivity Sets that promote or suppress selections and rules in the configuration model over time or depending on the end user.

You create Usages and Effectivity Sets in the Repository window and assign them to Models or Model nodes in the Model window. The Repository provides tools you can use to organize your Models and submodels in Folders.

When a Model is published to a production environment, it becomes the configuration model against which end users make selections to configure products and services. A Model publication is a copy of the Model that can be refreshed when the original Model undergoes revision and maintenance upgrades in Oracle Configurator Developer.

1.4.3 Building Models

The Oracle Configurator Developer Model window consists of the following modules which address different aspects of constructing and testing configurator functionality.

- The Model Window (see Chapter 4, "Constructing Model Structure")

- The Configuration Rules Module (see Chapter 5, "Constructing Configuration Rules")
- The User Interface Module (see Chapter 6, "Constructing the User Interface")
- The Test / Debug Module (see Chapter 7, "Testing and Debugging")

The Model, Configuration Rules, and User Interface modules present you with menus of commands and views of the configuration model you are building. The Test module launches a selected testing environment in which you can view a Model periodically during the development phase. Before viewing the Model in the test environment, you generate the Active Model and a User Interface to generate Model structure and rules and the graphical views necessary for users to interactively create configurations. The User Interface interfaces with the Active Model to provide access to customer requirements gathering, product selection, and customer-centric extensions.

There are two approaches to working in Oracle Configurator Developer:

- Self-contained mode
- Integrated mode

If you are working in the self-contained mode, you create the Item Master and build your Model, configuration rules, and User Interface entirely within Configurator Developer. You can chose to work this way if you are building a small-scale demo or prototype system.

Many real-world configuration models involve working in the integrated mode. In either mode, you build a configuration model based on product structure and item data. In integrated mode, end users configure products defined in Oracle Bills of Material (BOM), and pass completed product configurations on to Oracle Order Management. To build such a configuration model, you begin by importing BOM data into the CZ schema using concurrent programs. The *Oracle Configurator Implementation Guide* describes the data import process.

The BOM data import process creates a new BOM Model in Oracle Configurator Developer. You can then open this Model in the Model window to define additional Model structure, configuration rules, and a User Interface. If you need to add Model structure, for example to support guided buying or selling, you can also add Model structure nodes to the root node of the imported BOM Model.

After you have completed the Model structure, you create configuration rules that define how the parts of the Model are related to each other. After creating configuration rules, you use the **UI** module to generate a User Interface that reflects the structure of your Model. This generated interface enables you to test the

functionality of your Model and configuration rules in the **Test** module, without doing any additional User Interface development. When you are satisfied with the performance of Model and rules, you can customize the generated User Interface to meet your requirements. You can also generate and customize additional user interfaces to meet specific needs.

1.4.4 Deploying Models

The steps in the life cycle of a configuration model include creating the configuration model in Oracle Configurator Developer, thoroughly unit testing configurations, and publishing the Model to a system testing or production environment for use by test engineers or end users, respectively. The Oracle Configurator can be called from a variety of hosting applications, including Oracle Order Management, *i*Store, Telesales (Order Capture), as well as custom, Web-based applications.

Integration

If the Oracle Configurator is embedded in an Oracle Application such as Order Management or *i*Store, there may not be any additional set up required, depending on which options you select during the Oracle Applications RapidInstall process. However, certain profile options must be set in Oracle Applications to launch the embedded configurator. See the *Oracle Configurator Installation Guide* for more information.

If the Oracle Configurator is embedded in a custom Web application, the host application must generate the initialization and termination messages that start and stop the embedded configurator session. See the *Oracle Configurator Implementation Guide* for more information.

Testing

Before going live or into production with an Oracle Configurator, it is a good idea to system-test your Model for adequate performance, end user access, security, and any integration customizations. A configuration model itself should be periodically unit-tested while building it in Configurator Developer. However, because a Model may use **effective dates and Usages**, contain References to other models, or have multiple UI definitions, a single configuration model can provide many unique permutations depending on the end user or hosting application. Therefore, configuration models should be thoroughly tested in a separate environment using a variety of applications and effective dates to ensure the models function as intended.

Production

After several rounds of testing and updates in Configurator Developer, you can make a configuration model available to your end users. Test existing configurations against the new version before deploying it to ensure that users can restore previously saved configurations. If you made maintenance changes to the Configurator during the testing phase, you can easily bring the new Model on line without interrupting end user access. See Section 2.2, "Publishing" on page 2-11.

1.5 Multiple Language Support in Oracle Configurator

Multiple Language Support (MLS) enables you to create a Model and one or more User Interfaces in your base language and then display the runtime UI in any language in which you do business.

The Configurator Developer user interface, which includes all menus, field prompts, and dialog boxes, is available only in English. All text that the Oracle Configurator user enters appears in the language of the local machine's operating system. However, you can create Model structure and define configuration rules in the language installed on the machine running Configurator Developer, generate the UI, and then publish the Model in any language installed on your Oracle Applications database.

Warning: It is possible to change the language setting on your local machine after logging in to Configurator Developer and then modify existing models. However, Oracle Corporation strongly recommended that you do not do this, as it can corrupt Model data.

Oracle Inventory users enter alternate translations for item descriptions when defining items; Oracle Applications does not allow users to enter alternate translations for Inventory Item *names*. (You must use PL/SQL or a translation facility that you create to enter translations for non-BOM nodes. See the *Oracle Configurator Implementation Guide* for more information.) When you import a BOM Model, the concurrent program also imports these alternate translations into the CZ schema. At runtime, the host application specifies the language to use and displays the corresponding item descriptions in the Configurator window.

It is important to note that the text that is displayed at runtime may be different from the description of the BOM Standard Item or Option that appears in Configurator Developer. You determine how Configurator Developer creates UI

captions for BOM Standard Items and Options when you create a new UI. When using MLS, it is recommended that you generate UI captions using only item *descriptions* (this is the default). See Section 6.2, "Generating a New User Interface" on page 6-16.

When unit testing using the **Test** button, Configurator Developer displays the UI in the language of the local machine's operating system by default. When publishing the Model and UI, you determine in which languages the publication is available using the Language applicability parameter. See Section 2.2.2, "Publishing and Multiple Language Support" on page 2-24.

Warning: Do not use Configurator Developer to enter translations for non-BOM node names for a Model created in another language. Translating node names using this method corrupts the names when viewing the Model in the original language. Example: You create Model M1 in Configurator Developer in Japanese and then translate the non-BOM node names using Configurator Developer running in English on a different machine. When you open M1 in Configurator Developer running in Japanese, or view the Model in a Japanese runtime Configurator, the names will be corrupt.

Some additional setup and modifications are required before using Configurator Developer in an MLS environment. For more information, see the *Oracle Configurator Installation Guide* and the *Oracle Configurator Implementation Guide*.

2

Managing Models

Creating configuration models is an iterative process in which you create a Model, test and update it, and then retest it until the Model is approved for production use. Typically, a Model is tested under a variety of conditions to prepare it for the various ways in which it will be used by customers to configure products and services.

This chapter discusses some the powerful features of Configurator Developer that help you manage configuration models throughout the development, testing, and production phases.

2.1 Model Referencing

To reduce the time and effort required to create and maintain configuration models, you can create a **Reference** to use one Model as a subassembly in other Models. A Reference is also a type of node in Configurator Developer that indicates the location in the structure of a parent Model where another Model is referenced. You can create References manually in Configurator Developer, but you cannot create a Reference from a BOM Model to another BOM Model. Configurator Developer creates References automatically when you import a BOM Model that contains one or more other BOM Models. See Section 2.1.5, "References and BOM Models" on page 2-4.

For example, your organization sells many different styles of trucks and automobiles, but some of them use the same 200 horsepower, V6 engine. You can create and maintain one Model for this engine in Oracle Configurator Developer, and then simply create a Reference to that Model in all other Models (automobiles) that use it. Then, when the referenced Model is modified, the changes automatically propagate to all Models that refer to it.

Within the structure of a Model, a Reference node functions like a Component. Like a Component, a Reference node can be mandatory or optional; you can change its name, effectivity, and specify the minimum and maximum number of instances that are created at runtime. At runtime, each instance contains the entire structure of the referenced Model and is subject to all the rules defined in that Model.

A Reference node functions differently in an imported BOM Model. See Section 2.1.5, "References and BOM Models" on page 2-4.

When you work in a Model that contains a Reference, the structure of the referenced Model appears as a subtree of the parent Model. All of the referenced Model's attributes are read-only when viewed within the parent Model. See Section 2.1.8, "Display of Model References" on page 2-8.

2.1.1 References and Rules

You can refer to nodes in a referenced Model's structure when defining configuration rules for the parent Model. All of these rules, even those whose participant nodes are all within the referenced Model's structure, belong to and reside with the parent Model. You can create new rules for the parent Model, but all of the referenced Model's rules are read-only when viewed from the parent Model.

Use caution when using nodes within a referenced Model as participants in the parent Model's configuration rules. If a node in the referenced Model is modified or deleted, the rule in the parent Model becomes invalid and Configurator Developer displays an error message when you generate the **Active Model** for the parent.

For information about viewing rules in a referenced Model, see Section 2.1.8.1, "Rules" on page 2-8.

2.1.2 References and Effectivity

The root node of a Model is always effective. When a non-BOM Model is referenced by another Model, you can specify effective dates and assign an Effectivity Set and one or more Usages to the Reference node. For references to BOM Models, you can modify the Usages assigned to the Reference node. To modify the effective dates of a "nested" referenced Model (that is, a Reference within a Reference), you must open its parent in the Model window. For example, Model 1 references Model 2, and Model 2 references Model 3. To modify the effective dates of Model 3, you must open Model 2 in the Model window. See Section 3.3, "Effectivity" on page 3-4.

2.1.3 References and UI Definitions

A Model Reference does not specify anything about User Interface definitions within a referenced Model. Instead, the parent Model's UI definition contains a reference to the child Model's UI definition. This allows different parent UIs to refer to different UIs in a child Model.

Each time you generate a User Interface for a parent Model, Configurator Developer selects the child Model's most recently generated (or modified) UI definition. If a referenced Model has multiple UI definitions, you can select a different one after generating the UI for the parent Model. You can also prevent the child Model's UI from appearing at runtime by modifying its Visibility attribute. See Section 6.1.5, "Controlling Visibility" on page 6-10.

If a referenced Model does not have a UI definition, Configurator Developer creates a new one when you generate the parent Model's UI. The referenced Model's new UI resides in the referenced Model, not in the parent Model.

When you select a node within a referenced Model and then click the **Go To** button, Configurator Developer does *not* open the referenced Model in the Model window to display the location of the associated UI node. Instead, it simply selects the UI node to indicate its location within the parent Model's UI. Example: You are editing referenced Model M1-Test and Option 2 is selected. When you click on the **Go To** button, Configurator Developer opens Model M1-Test for editing in the Model window, expands the Model structure, and selects Option 2.

2.1.4 Copying Models with References

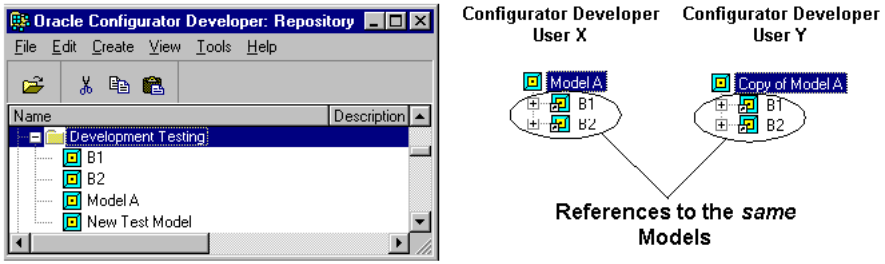
You can easily make copies of a Model, its structure, User Interface(s), and configuration rules in Configurator Developer. However, it is important to understand the effects of copying a Model that contains References to one or more Models. When you copy a Model in the Repository window and that Model contains a Reference, Configurator Developer does not create a new, separate copy of the referenced Model. When the operation is complete, both the original Model and the new copy refer to the *same* sub-Model. As a result, any changes in the referenced Model affect not just the original Model, but the copied Model as well.

Example of Copying a Model with References

Model A contains references to B1 and B2. You copy Model A and rename it "New Test Model." Both Model A and New Test Model now refer to Model B1 and Model B2. Any changes made to B1 or B2 will affect both Model A and New Test Model.

Figure 2–1 shows how the Models in this example appear in the Repository and Model windows after the copy operation.

Figure 2–1 Copying a Model with References



Note: You can programmatically copy a parent Model and all of its child (Referenced) Models using the PL/SQL package `CZ_modelOperations_pub`. This package contains a set of APIs that automate many of the tasks required to maintain configuration models. For more information, see the *Oracle Configurator Implementation Guide*.

2.1.5 References and BOM Models

One important use of References is to represent the relationship between a BOM Model that contains other Models when you import it into Configurator Developer. When you import a BOM Model that contains one or more ATO or PTO Models, Configurator Developer creates a Reference node for each child Model to make it easy to locate them in the parent's structure. You can assign one or more Usages to the node of a BOM Model that is referenced and control whether it appears in the runtime UI (see Section 6.1.5 on page 6-10), but all other Reference node attributes are read-only.

If a BOM Model exists within other BOM Models (for example, an ATO Model within a PTO Model), it is not imported into Configurator Developer multiple times. The import concurrent program imports the BOM only once and creates references to it from the other BOM Models in which it is included as a component.

This allows the rules and UI for the referenced Model to be maintained in one place and prevents any duplicate BOMs from existing in the database.

You can create a reference from a Configurator Developer Model (non-BOM) to a BOM Model, but Developer does not allow you to manually create a reference from a BOM Model to another BOM Model. This is because all imported BOM Models must retain their structure as defined in Oracle Bills of Material. If a BOM Model's structure could be altered in Configurator Developer (by referencing another BOM, for example), the resulting Model would not be orderable from Oracle Order Management.

Although an imported BOM Model can contain one or more other ATO or PTO Models (and these sub-Models appear as References in Configurator Developer), this structure can only be created and maintained in Oracle Bills of Material. In other words, you cannot manually create a Reference from a BOM Model to another BOM Model in Configurator Developer. See Section 2.1.7, "Creating a Model Reference" on page 2-7.

Additionally, if a Configurator Developer Model references a BOM Model, you cannot create a Reference from another BOM Model to that Configurator Developer Model. For example, a non-BOM Model called M1 references BOM Model B1. You cannot create a Reference from another BOM Model to M1, since this would create a Model in which a BOM Model contains another BOM Model.

2.1.5.1 References and Optional BOM Models

A referenced BOM Model is not required to create a valid configuration if the **Optional** check box is selected for that Model in Oracle Bills of Material. If an optional BOM Model contains a Configurator Developer Feature that is required, unintended results may occur at runtime. (List of Options Features that have both the **Minimum** and **Maximum Number of Selections** set to 1 are considered required Features.) This can occur when an optional BOM Model has a required Feature:

- At the Model level
- In a child (referenced) BOM Model
- In a referenced Configurator Developer Model

Example: Model A references Model B and Model C, which are both optional BOM Models. Model B and C are variations of the same component, and a valid configuration may contain one and only one of these components. Both Models contain a required Feature, and reference a Configurator Developer Model that contains a required Feature (this structure is shown in Figure 2–2 on page 2-6).

At runtime, the end user selects and configures Model B, satisfying the configuration requirement for that component. However, a selection from Model C is still required because it contains a required Feature. In this situation, the end user will not be able to save a valid configuration of Model A, since only Model B *or* Model C is allowed.

To avoid this situation, do the following:

- Make any required Features optional by changing the **Minimum Number of Selections** to 0.
- If the end user must make a selection from the Features to create a valid configuration, create a Logic rule that uses the Requires relation to tie the Features to the parent BOM Model.

  For example, using the Model shown in Figure 2–2 on page 2-6, change the **Minimum** for all of the required Features to 0, and then create the following rules:

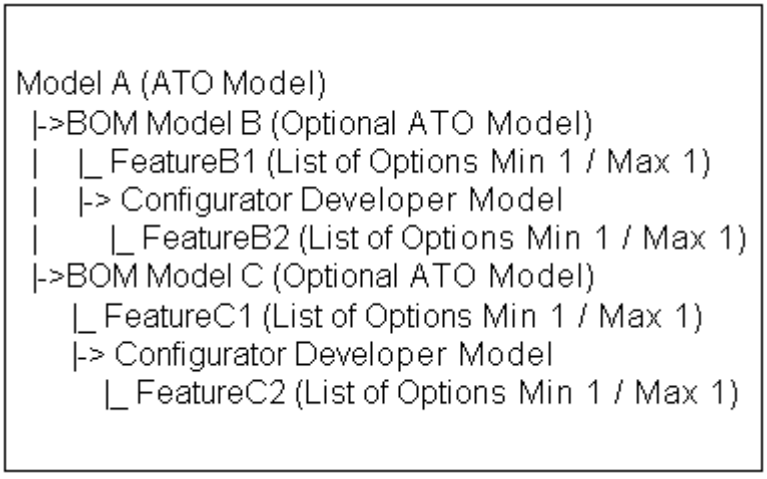
  "BOM Model B REQUIRES FeatureB1"

  "BOM Model B REQUIRES FeatureB2"

  "BOM Model C REQUIRES FeatureC1"

  "BOM Model C REQUIRES FeatureC2"

Figure 2–2 Optional BOM Models with Required Features



2.1.6 Updating Referenced Models

When a Model is the target of a Reference and you make any changes to the structure of that Model, you may need to update the parent Model. This is because:

- Rules in the parent may refer to deleted structure nodes in the child.
- Newly added Options in the child may be missing from Explicit Compatibility rules in the parent.

Note that the logic generated for Property-based Compatibility rules in the parent Model that refer to Features in the child Model may change when you modify Properties in the child. Therefore, it is a good idea to use Property-based Compatibility rules whenever possible, as this type of rule reduces the amount of time and effort required to maintain configuration rules.

Configurator Developer does not automatically update the parent Model when you modify the structure of a referenced Model. You may be able to resolve any potential problems with the configuration simply by regenerating the Active Model for the parent Model. However, if the child Model's logic is not up to date, Configurator Developer displays a message when you generate the Active Model for the parent. In this case, you must regenerate logic for the referenced Model before you can successfully create the Active Model for the parent. You can then re-import or make change to those Models as required.

Note that not all changes to Models that are referenced require you to update their parent Models. For example, modifications to the configuration rules or UI definitions of a referenced Model do not require changes within the parent, except perhaps for re-testing.

You cannot delete a Model if it is referenced by another Model.

2.1.7 Creating a Model Reference

There are two ways to create a Model Reference:

- Create it manually in the Model window
- Import a BOM Model

To manually create a reference, choose New Model Reference from the Create menu in the Model window, or right click using the mouse and select the Model to reference from a list. If a parent Model's root node is a Component, you cannot create a Reference to a Model that has a Product node as its root node.

When you import a BOM Model, Configurator Developer creates a configuration model in the Repository for the root BOM Model and for any BOM Models that it

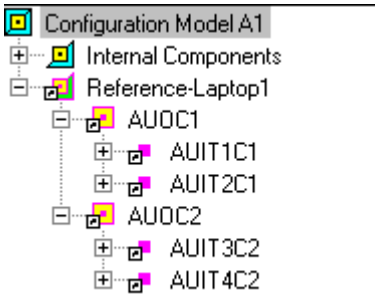
contains. Configurator Developer also creates the Model References that link this structure together within each parent configuration model.

Note that running the Refresh All Configuration Models concurrent program may cause additional Models and References to be created (if BOM structure has been added) or modifications to or the deletion of existing Model references.

2.1.8 Display of Model References

The entire structure of a referenced Model is viewable as if it were part of the structure of its parent. You can tell the referenced structure from the parent Model's structure by its appearance. Nodes that represent BOM Model references and their child nodes are indicated by a small arrow.

Example 2–1 Model Reference Nodes



2.1.8.1 Rules

You can view all of the rules that apply to a referenced Model from its parent Model, but the referenced Model's rules are read only. When you select a Reference Model node in the Model window and then go to the Configuration Rules module, the Configuration Rules view lists all of the rules for the root Model as well as all rules associated with the referenced Model. For example, if the referenced Model name is Test-M1, the name of the rules folder for that Model is "Test-M1 Rules". Only one folder appears in the Configuration Rules view for the referenced Model, even if there are multiple references to it in the parent Model.

2.1.8.2 Renaming Reference Nodes

The names of Models and Reference nodes can be confusing since you can easily rename Reference nodes and root-level Model nodes in Configurator Developer. As

a result, the name of a Reference node can be different than the name of the Model to which it refers. Configurator Developer always maintains the original name of the Model, even if you change the name of its Model node in the Repository window. For BOM Models, the original name is the Oracle Inventory Item name followed by the inventory organization ID and Item ID. For non-BOM Models, the original name is the name you entered when creating the Model in the Repository.

For example, you create a Reference to Model MCS-Alpha 1 within the Envoy Custom Laptop Model. The default name of the Reference node is "MCS-Alpha 1" but you rename the node to "Ref to Proto-9 Subassembly". Configurator Developer updates the name of the Reference node in the Attributes view, but "MCS-Alpha 1" still appears below the new name. You can click the **Go To** button in the Attributes view to open Model MCS-Alpha 1 for editing in the Model window, or switch to the Repository window to view the MCS-Alpha 1 Model node.

Note: If you change the default name of a Reference node in Configurator Developer, the name of that item in the Summary section of the Java applet or the Summary screen in DHTML does not match the name that appears in the host application when the end user saves the configuration.

2.1.8.3 User Interface

You can view a referenced Model's UI in the parent Model's UI, but the referenced Model's UI is read only. Only the pages and their contents from referenced UIs are displayed in the Model window's UI tree view. (Component trees from referenced UIs do not appear at runtime.) In the UI module, the Model tree also displays a referenced Model's structure this way. See Section 6.1.8, "Referencing and the Model User Interface" on page 6-15.

When you select a Reference node, an Associated UI list displays all UI objects representing the selected node in the context of the top-level Model that is open for editing. Configurator Developer indicates UI objects defined within referenced Models with the standard reference icon for that UI node type. Use the **Go To** button to navigate to any of these UI nodes in the Model window.

All pages of a referenced UI appear in read-only mode when viewing that UI in the parent Model. You can also open any page of a referenced UI in the Preview window (also in read only mode).

The **Go To** button functions differently when you select node within a referenced Model from the list of associated UI nodes. When you do this, clicking the **Go To** button displays the selected UI node within the UI of the Model currently open for

editing; the system does not open the referenced Model in the Model window. See Section 9.20.1, "Associated UI Nodes" on page 9-41.

2.1.8.4 Viewing References

You can view all of a Model's references in the Repository window by selecting the Model and then choosing **View > References**. Then, in the Model References window, select a Model and click **Go To** to open it for editing in the Model window.

2.1.9 Operations on a Model Reference

You can perform the following operations on a Reference node in Configurator Developer:

- Rename it
- Change its effective dates and Usages
- Delete it (this deletes the entire Reference, with all its rules and UIs)
- Modify the **Minimum** and **Maximum Instances** attribute, but only if the Reference is to:
  - A Configurator Developer Model
  - An ATO or a PTO BOM Model that is a direct child of a PTO BOM Model

  To modify a Reference's **Instances** attribute, its parent Model must be open for editing in the Model window. For more information about the **Instances** attribute, see Section 3.5, "Multiple Instantiation in Solution Models" on page 3-14.

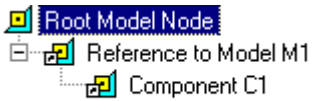
When editing a Model that contains one or more References to other Models, you cannot modify or delete:

- The referenced Model's structure or any attributes of node's within the structure
- Any of the referenced Model's configuration rules
- Any of the referenced Model's UI definitions

Structure nodes that belong to a Reference can participate in rules as if they were part of the parent Model's structure. In addition, the min / max number of **Instances** on the root node of the Reference can participate in Numeric rules in the same way as min / max number of **Instances** on a Component node.

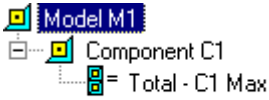
However, Numeric rules cannot contribute to sub-components in the referenced Model. For example, a Numeric rule that you create in Root Model cannot contribute to Component C1, as shown in Figure 2–3.

Figure 2–3 Numeric Rules and Referenced Components



However, you can overcome this limitation by opening Model M1 for editing in the Model window and creating a Total that contributes to Component C1, as shown in Figure 2–4.

Figure 2–4 Using a Total to Contribute to a Referenced Component



2.2 Publishing

Publishing is typically the final phase of the configuration model development process. When you create a configuration model in Oracle Configurator Developer, you need to unit test, modify, and retest it before making it available to users in either a system testing or production environment. Publishing is a process that creates a copy of a configuration model's structure, rules, and UI on a specific database to make it available for testing or production use from different hosting applications.

Creating a **publication** is the first step in the publishing process. A publication is a unique deployment of a configuration model that enables you to control its availability when invoked by a hosting application and the UI that is displayed to the end user. A configuration model can have multiple User Interfaces and you can create many publications for the same Model. However, a publication corresponds to only one configuration model and User Interface.

Creating a publication is a two step process:

1. Define a **source publication** in the Model Publishing window in Configurator Developer. This a new record in the database on which Configurator Developer is running. (See Section 2.2.1.1 on page 2-14.)

   In this step you define the publication's:

   - **Attributes**, which include the Model, a User Interface, and the database where the publication will be available to hosting applications.
   - **Applicability parameters**, which include Usages, an effective date range, a publication Mode, and one or more languages.

2. Create a **remote publication** by running a concurrent program in Oracle Applications. The concurrent program copies the publication information, configuration model data, and UI to the target database you specified in step 1. (See Section 2.2.1.4 on page 2-22.)

   Note: The publication cannot be viewed in a Configurator window until the publication concurrent program completes successfully.

When an end user hosting launches Oracle Configurator from a hosting application, the Configurator window searches the database for the publication whose definition matches the information sent by the hosting application. If no matching publication is found and the Model was created from an imported BOM Model, the Configurator window displays the BOM Model in the Java applet (the Java applet is described in Section 6.1.2.1 on page 6-3). If no matching publication is found and the Model was created in Configurator Developer, the Configurator window displays an error.

See the *Oracle Configurator Implementation Guide* for:

- Things to consider when planning for publishing
- Details about how a hosting application selects a publication
- A description of the database tables used during the publishing process
- Examples of how to maintain publications once they are available in your production environment

2.2.1 The Publishing Process

Following is an overview of the publishing process:

1. Understand the publication process and carefully plan for how your Model publications will be used. (Refer to the section on planning publications in the *Oracle Configurator Implementation Guide*.)
2. Using Oracle Configurator Developer, create a new source publication. See Section 2.2.1.1 on page 2-14.
3. In Oracle Applications, submit one of the publication concurrent programs to copy all related Model data (UI, product structure, and rules) to the database you specified when creating the source publication. See Section 2.2.1.4 on page 2-22.
4. Test the publication by invoking the Model from a hosting application such as Oracle Order Management, *i*Store, or TeleSales to ensure that it functions as intended.
5. Use Oracle Configurator Developer to make any necessary changes to the Model, rules, UI, or the publication's definition.
6. Republish the Model so any changes to the Model structure, rules, or UI are visible to end users. See Section 2.2.1.3 on page 2-20.

   Note: It may also be necessary to synchronize publication data for configuration models that are based on imported BOM Models. For more information about data synchronization, see the *Oracle Configurator Implementation Guide*.

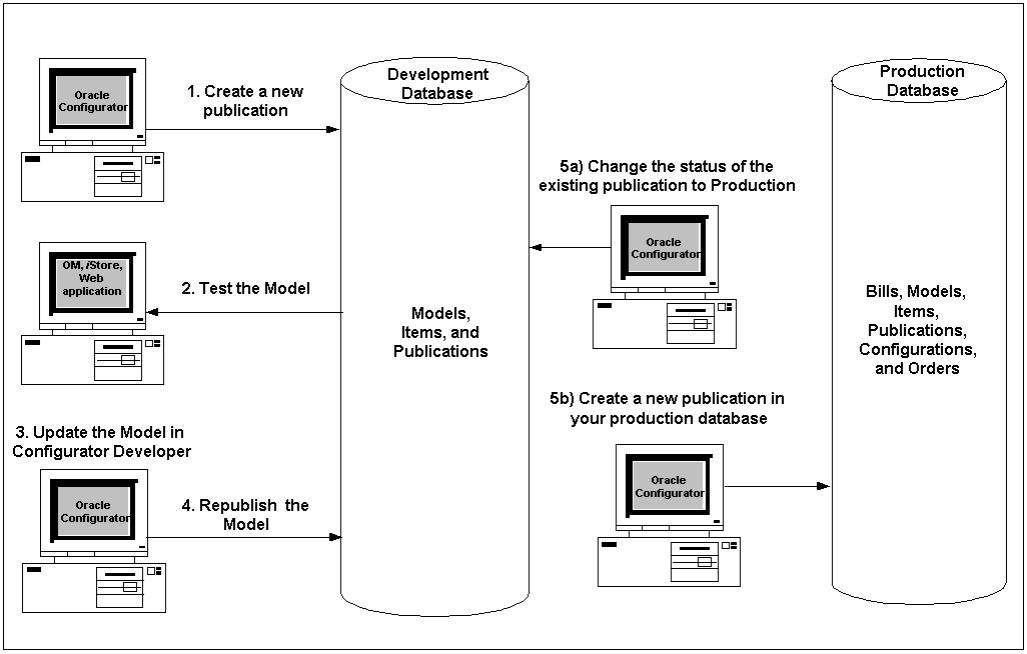
7. When testing is complete, make the Model available for production use.

   If you use the same database for both development and production activities, you can do this by changing the Mode applicability parameter of the existing publication from Test to Production (see step 5a in Figure 2–5, "Example of the Publication Process").

   If you maintain separate development and production environments, create a new publication in your production database. (This is shown in step 5b in Figure 2–5 on page 2-14). If your system is set up correctly, you can do this by selecting your production database from a list in the Model Publishing window.

   For an example of how a sample organization maintains its Model publications, see the *Oracle Configurator Implementation Guide*.

Figure 2–5 Example of the Publication Process



For details about the database tables used when publishing Models, see the *Oracle Configurator Implementation Guide.*

2.2.1.1 Creating a New Publication

Use the procedure described in this section when the Model structure, its rules, or UI has changed in Configurator Developer since it was last published. If you want to publish the same configuration model but specify different applicability parameters, you can create a copy of an existing publication, and then modify its applicability parameters. This process is described in Section 2.2.1.2 on page 2-19.

Before you can create a new publication, the UI and Active Model for the configuration model must be up to date in Configurator Developer. A configuration model is not up to date if you changed the Model structure since the last time the User Interface was generated. In this case, you must either refresh the UI or create a new one before publishing the Model. You must also regenerate the Active Model

before publishing the Model if you changed any configuration rules. See Section 5.1, "The Configuration Rules Module" on page 5-1.

For an overview of the publishing process, see Section 2.2 on page 2-11.

Note: Although it is possible to create and maintain publications on separate databases (for example, a development and a production instance), configuration models that will be available to customers should be *published* only from a single development environment. For more information, see the *Oracle Configurator Implementation Guide*.

To create a new publication:

1. In Configurator Developer, generate the Active Model and refresh the UI for the Model you want to publish.
2. In the Repository window, choose **Tools > Publishing**. The Model Publishing window appears. (For details about this window, see Section 8.11.1 on page 8-10.)
3. Click **New**. In the New Publication dialog, click the list of values button (**...**) for the **Model** field and select the Model to publish.

   The list of values shows Model names as they appear in the Repository window. You must enter the Model to publish using this name, even though this name may not be the same as the root node in the Model window. (For more information, see Section 8.13, "Tree Views" on page 8-12.)
4. Click **OK**.
5. Select a **User Interface** from the list, or accept the default value of **None** if you do not want to associate a UI with this publication.
6. Select to which **Database Instance** you want the publication to be available to hosting applications. (This is referred to elsewhere as the "target" database.)

   The list includes all Oracle Applications database instances defined in your organization, not just the instances that are currently enabled. (Database maintenance tasks are described in the *Oracle Configurator Implementation Guide*.)

Note: If you are using Multiple Language Support (MLS), all installed languages on the source and target databases must be the same when publishing to a remote server. For more information about MLS and publishing, Section 2.2.2 on page 2-24.

Note: When you select a Model, Configurator Developer provides a default value for the **Product ID** field. For information about how Configurator Developer determines this value, see the *Oracle Configurator Implementation Guide*.

7. Enter applicability parameters for this publication, including a **Mode**, one or more **Applications**, **Usages**, and effective dates. (For details about each parameter, see Section 2.2.1.1.1 on page 2-16.)
8. Click **OK**. Developer displays an error message if one or more applicability parameters overlaps with an existing publication. If this occurs, modify the parameters as required, then click **OK**. The new publication appears in the list of publications with a unique **ID** number and a **Status** of Pending.

Note: Make a note of the number that appears in the **ID** column. You need to enter this value if you choose to run the Process a Single Publication concurrent program in Oracle Applications to copy the publication to the target database.

9. Click **Close** to close the Model Publishing window.
10. Log in to Oracle Applications using the Configurator Developer responsibility.
11. Copy the Model and publication data to the database you specified in step 5. See Section 2.2.1.4 on page 2-22.

2.2.1.1.1 Applicability Parameters You specify applicability parameters when defining a publication to control its availability to hosting applications. When an end user invokes Oracle Configurator, the hosting application creates a **session initialization message.** Parameters in this message request the specific publication to display, and must match the applicability parameters defined for the publication to do so. For more information about the session initialization message, see the *Oracle Configurator Implementation Guide*.

You can create multiple publications for the same configuration model on the same database, but each publication's applicability parameters must be unique. In other words, publications with *overlapping* applicability parameters cannot exist on the same database. For details, see Section 2.2.1.1.2 on page 2-18.

Applicability parameters determine which publication to display based on the following criteria:

- **Mode**: Use this parameter to specify whether the publication is used for production or testing activities, or if it is unavailable. Values include **Test**, **Production**, and **Disabled**. Only publications with a mode of **Production** or **Test** can be accessed by a hosting application such as Order Management or *i*Store. A publication marked as **Disabled** is unavailable to hosting applications, but is *not* deleted from the database.
- **Valid From/Valid To**: Use this region to indicate that the publication is available only for a specific period of time. (At runtime, this time period is compared to the system date of the database in which the publication exists.) Valid dates may range from 01/01/1601 through 12/31/4710 (inclusive).

  The dates and times you specify here determine only whether the *publication* is available when accessed by a hosting application; the effective dates (or Effectivity Set) defined within the Model itself determine whether parts of the Model structure, configuration rules, and UIs are available at runtime.

  Note: At runtime, the **Valid From/Valid To** time period is compared to the system date of the target database in which the publication exists. If the machine on which the source publication was created is out of sync with the time of the machine on which the hosting application is running, the Model may not appear as expected. For example, a Java applet UI appears instead of the DHTML window. To resolve this issue, override the default **Valid From** setting by selecting the **No Start Date** box.

- **Usages**: Use Usages as an additional criterion to control whether a publication is available when invoked by a hosting application. To limit a publication's availability based on Usages, select **Publish for selected usages** in the Model Publishing window, and then enter one or more Usages (or select from the list of values). Select **Publish for any usage** if you do not want to limit the publication's availability based on Usages. For an overview of Usages, see Section 3.3.2 on page 3-6. For an example of how you can use Usages to limit a publication's availability, see the *Oracle Configurator Implementation Guide*.

- **Applications**: Use this parameter to specify which hosting applications can access the publication. For example, you can specify that a publication is available to Oracle Order Management and *i*Store, but not TeleSales. This list displays the short name of all applications registered in Oracle Applications. For more information about this parameter, see the chapter on session initialization in the *Oracle Configurator Implementation Guide*.

  For information about registering applications and defining application short names, see the *Oracle Applications System Administrator's Guide*.

  For a complete list of applications that integrate with Oracle Configurator, as well as each application's short name, see the *Oracle Configurator Release Notes*.

- **Languages**: Use this parameter to specify in which languages the publication is available. You must select at least one language. The list of values includes the base language and all installed languages on the Oracle Applications database on which Configurator Developer is running. For more information, see Section 2.2.2, "Publishing and Multiple Language Support" on page 2-24.

  You can view a publication's applicability parameters in the Model Publishing window. For more information, see Section 8.11.1 on page 8-10.

2.2.1.1.2 Overlapping Applicability Parameters Only one publication can exist on the same database for the same product, publication mode, application, Usage, and so on. If other publications currently exist on the target database, at least one applicability parameter must be unique to create the new publication. Oracle Configurator Developer ensures that any changes you make to existing publications will not create conflicting applicability parameters.

Table 2–1 provides examples of how Configurator Developer does not allow multiple publications for the same Model on the same target database instance.

Table 2–1 Comparing Applicability Parameters

Applicability Parameter	Publication A	Publication B	Publication C	Publication D
Publication Mode	Test	Test	Test	Test
Applications	OM	*i*Store, Order Capture, OM	OM	OM
Usage	Desktop_A1	Desktop_A1	Desktop_A1	LaptopPC
Date Range	10-JUN-00 to 01-DEC-00	10-JUN-00 to 01-DEC-00	10-OCT-00 to 10-DEC-00	10-JUN-00 to 01-DEC-00

In this example, the configuration model is successfully published and the result is Publication A. When you compare this publication with requests to create publications B, C, and D you can see that:

- The request to create Publication B fails because although the publication is available to *i*Store and Order Capture, it is also available to Order Management (OM). Therefore, the Applications parameters overlap.
- The request to create Publication C also fails because the date ranges overlap.
- The request to create Publication D is successful because the Usage specified is unique, even though all of the other applicability parameters are the same as Publication A.

For more information about maintaining publications, see the *Oracle Configurator Implementation Guide*.

2.2.1.2 Copying an Existing Publication

Use the **New Copy** button in the Model Publishing window to create a new publication by copying an existing publication's attributes and applicability parameters. You may want to do this if, for example, you want to create additional publications with many of the same attributes as an existing publication. By copying an existing publication, you do not have to re-enter the Model, UI, target database, and so on.

> **Note:** The publication's applicability parameters must be unique before you can create it. For details, see Section 2.2.1.1.2 on page 2-18.

To copy an existing publication:

1. In the Model Publishing window, select a publication from the list of existing publications.
2. Click **New Copy**.
3. In the Copy Publication window, modify the new publication's applicability parameters as required. (Applicability parameters are described in Section 2.2.1.1.1 on page 2-16.)
4. Click **OK**. Make a note of the publication ID for future reference.
5. Run one of the publication concurrent programs to copy the publication and Model data to the target database. See Section 2.2.1.4 on page 2-22.

2.2.1.3 Republishing

When you make changes to a Model in Configurator Developer, the changes do not affect the published version of that Model. To update the published Model, you must copy all new and modified data to the database by **republishing** the Model. A republished Model is not available for testing or production activities until an Oracle Applications concurrent program successfully copies the new or modified Model data to the target database. (This is described in Section 2.2.1.4 on page 2-22.)

You cannot modify applicability parameters when republishing a Model, but you can modify these parameters once the publication's status is Complete. See Section 2.2.1.5 on page 2-22.

> **Note:** If you publish a Model and later modify its structure or rules in Configurator Developer, you can republish the Model using the same applicability parameters if no other publications exist for the same Model and applicability parameters. If there is another publication for the same Model with the same applicability parameters, you must either delete the existing publication before republishing the Model, or change the existing publication's applicability parameters so they do not overlap.

When you select a publication and then click **Republish** in the Model Publishing window, Configurator Developer creates a new publication record, adds a new publication record ID at the end of the existing publication, and changes its status to Pending Update. The new publication record ID that is appended to the existing record indicates that the two records are related.

For example, publication record 1001 exists for Model M1 and its status is Complete. Table 2–2 shows how Configurator Developer creates a new record (ID # 1002) in the Model Publishing window when you republish Model M1.

Table 2–2 Republishing a Model

ID	Model	UI	Published	Status
1001	M1	UI-1	11/23/2000 12:01	Pending Update {1002}
1002	M1	UI-1	11/25/2000 14:35	Pending

If you decide you do not want to create the publication, you can delete the Pending process (ID 1002) before its status changes to Complete. See Section 2.2.1.6 on page 2-23.

Table 2–3 shows how the new publication record's status changes to Processing when the Oracle Applications concurrent program selects it.

Table 2–3 Processing a Republish Concurrent Request

ID	Model	UI	Published	Status
1001	M1	UI-1	11/23/2000 12:01	Pending Update {1002}
1002	M1	UI-1	11/25/2000 14:35	Processing

Table 2–4 shows how Configurator Developer updates the publication records when the concurrent program successfully copies the Model data to the target database. The status of the new record changes to Complete and the old record (ID 1001) no longer appears.

Table 2–4 Updating Status when Republish is Successful

ID	Model	UI	Published	Status
1002	M1	UI-1	11/25/2000 14:35	Complete

Table 2–5 shows how Configurator Developer updates the publication records when the concurrent program fails. The status of the original publication reverts to Complete and the new record is set to Error.

Table 2–5 Updating Status when Republish Fails

ID	Model	UI	Published	Status
1001	M1	UI-1	11/23/2000 12:01	Complete
1002	M1	UI-1	11/25/2000 14:35	Error

You must delete the record that failed by selecting it in the Model Publishing window and then clicking Delete. See Section 2.2.1.6, "Deleting a Publication" on page 2-23.

To update a publication:

1. Choose **Tools > Publishing** from the Repository window.
2. Select the publication to update.

3. Click **Republish**. Configurator Developer creates a concurrent request to copy Model data to the specified database. Make a note of the new publication ID for future reference.
4. Click **Close**.
5. Copy the Model data to the database. See Section 2.2.1.4, "Copying Model Data to a Database" on page 2-22.

2.2.1.4 Copying Model Data to a Database

When you create a new publication, Configurator Developer inserts a new source publication record in your development database. Before a hosting application can access a publication, the Model data must be copied to the target database. You do this by running one of the publishing **concurrent programs** in Oracle Applications. There are two publication concurrent programs: one selects only a single source publication that you specify, while the other selects all source publications that have a status of Pending.

The publication concurrent program that you run select the source publication record(s) in the development database and copies the publication, Model, and User Interface data to the target database. When the program completes successfully, it creates a new remote publication record in the target database. At this point, the status of the publication changes to Complete in the Model Publishing window, and hosting applications can invoke the configuration model in a runtime Oracle Configurator.

For more information about the publication concurrent programs, see the *Oracle Configurator Implementation Guide*.

Note: If you make any changes to the Model or any of its referenced Models in Configurator Developer before the concurrent request copies the data to the target database, the Active Model will not be up to date and the concurrent request will fail. If this occurs, Configurator Developer sets the publication status to Error. To create the publication and change its status to Complete, you must republish the Model. See Section 2.1.6, "Updating Referenced Models" on page 2-7 and Section 2.2.1.5 on page 2-22.

2.2.1.5 Editing a Publication

Edit a publication's applicability parameters when you want to change its availability to hosting applications. For example, you can add hosting applications

to the **Applications** parameter to make it more widely available, modify the effective dates, or change the publication **Mode**. (To make a publication available for a different application or Usage *and* for a different date range, you must create a new publication. See Section 2.2.1.1 on page 2-14.)

You can edit a publication only if its status is Complete or Pending. For more information about publication statuses, see the *Oracle Configurator Implementation Guide*.

Changing a publication's applicability parameters only affects its availability to hosting applications, it does not create a new publication in the database, copy Model data, or modify existing Models in any way.

Configurator Developer does not allow you to make changes that would create publications with overlapping applicability parameters in the same database instance. Overlapping applicability parameters are explained in Section 2.2.1.1.2 on page 2-18.

To modify a publication's applicability parameters:

1. Choose **Tools > Publishing** from the Repository window.
2. Select the publication to modify, then click **Edit**.
3. Modify the applicability parameters. You can modify the mode, applications, Usages, and effective dates.
4. Click **OK**.

Note: It is not necessary to run one of the publication concurrent programs after updating a publication's applicability parameters. This is because the changes propagate automatically to the remote publication in the target database.

2.2.1.6 Deleting a Publication

Deleting a publication prevents any runtime Configurators from accessing the Model for any further testing or production activities. You may also want to delete a publication if, for example, one of the publication concurrent programs has an error status upon completion, and you want to republish it.

Deleting a publication removes the publication record from the database, but it does not delete any Model data in the CZ schema that is associated with the publication.

For information about purging publication records and Model data from a database, see the *Oracle Configurator Implementation Guide*.

You can also make a publication unavailable by setting the **Mode** applicability parameter to **Disabled**. You might want to disable a publication if, for example, you need to make it temporarily unavailable. You can then make the publication available again by changing the **Mode** to either **Test** or **Production**. (See Section 2.2.1.5, "Editing a Publication" on page 2-22.)

To delete a publication, its status must be Complete, Pending, or Error.

To delete a publication:

1. Choose **Tools > Publishing** from the Repository window.
2. Select the publication you want to delete, then click **Delete**.
3. Click **OK**.

Note: It is not necessary to run one of the publication concurrent programs after you delete the source publication as described above. Deleting a source publication automatically deletes the remote publication in the target database.

2.2.2 Publishing and Multiple Language Support

Multiple Language Support (MLS) enables you to create configuration models and one or more User Interfaces in your base language, and then display the Model in any language in which you do business. (For more information about MLS, see the *Oracle Configurator Implementation Guide*.)

When you create a new publication, any alternate language information for the Model is exported to the target database (in other words, translations of node names, descriptions, and so on). The Language applicability parameter determines in which languages the publication can be displayed at runtime. For example, the host application indicates that the desired language is French. If the publication's Language applicability parameter includes French, the configuration model appears in the Oracle Configurator window in that language (of course, all of the other applicability parameters must match as well).

Once you have published a Model, you can enable or disable languages for a publication by modifying the publication's Language applicability parameter. For example, a publication's Language applicability parameter includes German, French, and English. You edit the publication in the Model Publishing window,

deselect English, and then click **OK**. As a result, the publication is no longer available when, for example, an Oracle Order Management user requests the publication in English (in this case, the host application displays an error). Because it is available in only German or Spanish, the hosting application must request the publication in one of these languages to access it in a Configurator window. (Editing a publication is explained in Section 2.2.1.5 on page 2-22.)

Republishing is required only if changes were made to the Model or UI in Configurator Developer or if the BOM Model or any Inventory Items were modified in Oracle Applications. Example: Additional translations are entered for existing items in Oracle Inventory and new items are added to the BOM. You must reimport or refresh the BOM and then republish the Model so the changes appear when the publication is accessed by a host application. Republishing is described in Section 2.2.1.5 on page 2-22.

For more information, see Section 1.5, "Multiple Language Support in Oracle Configurator" on page 1-10.

3

About Models

This chapter describes the different types of Models you can build in Configurator Developer.

For information about designing Models to optimize the performance of your runtime Oracle Configurator, see the *Oracle Configurator Performance Guide*.

3.1 The Item Master

The runtime Oracle Configurator and Oracle Configurator Developer use the Oracle Configurator schema (CZ schema) within the Oracle Applications database to access and store data. The CZ schema stores product data in the **Item Master**, which is a subschema within the CZ schema. When building configuration models in Oracle Configurator Developer, you use data in the Item Master. The items in the Item Master are either created from scratch in Configurator Developer, or are imported. Imported data is used only for defining the configuration model.

Legacy product data, such as Bills of Material or Pricing information, can be imported into the Item Master. Generally, the data source is either an Oracle Applications database or a non-Oracle Applications database. For consistency, imported data should be maintained in the source database.

In most development situations, Item Master data is imported. See the *Oracle Configurator Implementation Guide*.

Note: Do not confuse the Oracle Configurator Developer Item Master with the Oracle Applications Item Master. In this user's guide, the term Item Master always refers to the Configurator Developer Item Master, unless otherwise indicated.

The Item Master consists of **Items**, which are specific elements of a product, and **Item Types**, which are logical groupings of Items. For more information about these entities, see Section 3.4.1 on page 3-8.

Oracle Configurator Developer provides a mechanism called **Populators** to link data in the Item Master to the Model. Populators allow you to quickly update your Model to reflect changes in product data. For more information about Populators, see Section 4.4 on page 4-8.

You can change the way data appears in the Item Master by choosing options from the **View** menu. See Section 9.8 on page 9-11.

To create a report on the content of the Item Master, generate a Model Report. See Section 9.5 on page 9-5.

3.2 Properties

Model nodes can have **Properties**. A node's Property has a specific value that describes an instance of that node, but is not itself something that an end user can select in a runtime Oracle Configurator. Examples of common Properties are weight, diameter, and voltage. The values of Properties can also be used when defining configuration rules. See Section 5.9, "Comparison Rules" on page 5-29 and Section 5.10.1, "Property-based Compatibilities" on page 5-32.

To import Properties as part of an imported BOM Model, the Properties must be defined as Descriptive Elements in an Item Catalog Group associated with the BOM in Oracle Bills of Material (see Section 3.4.1 on page 3-8).

Note: Because Properties can be used when defining some types of configuration rules, it is important to consider whether you want Descriptive Element Values to be imported as strings or as numbers. Use the database setting ResolvePropertyDataType to control how Descriptive Element Values are interpreted when importing BOM Models. For more information on this database setting, see the *Oracle Configurator Implementation Guide*.

For the portions of your Model that you create in Oracle Configurator Developer, you can add, delete, and edit Properties directly on nodes of the Model, or you can add Properties to Items in the Item Master. When you add or delete a Property of a node in the Model, you affect only the selected node. This is different from Properties in the Item Master, where additions and deletions affect all Items of the

selected Item Type. For more information about the Item Master, see Section 3.1 on page 3-1.

You can also add Properties to imported nodes, including BOM Models, BOM Option Classes, and BOM Standard Items. However, the following restrictions apply:

- The new Property cannot have the same name as one that already exists on an Item or Item Type in the Item Master.
- You cannot add Properties to a node that is a Reference to a BOM Model or to any of the nodes within the referenced structure.

If required, you can also edit or delete any Properties that you add to an imported BOM node, as long as the node is not a Reference or part of a referenced Model's structure. However, all Properties that are imported with the BOM are read-only and cannot be modified or deleted in Configurator Developer.

When you use an Item or Item Type in your Item Master to create a node in your Model (for example, by using a Populator), the Model node always reflects all the currently defined Properties and Property values of the originating node in the Item Master. These inherited Properties cannot be deleted or edited in your Model. Properties that you define directly in your Model do not affect the Item Master.

Section 9.14.7, "Properties" on page 9-30 describes how to add, edit, and delete Properties. Section 9.9.1, "Manage Properties" on page 9-13 describes how to work with Properties on a project-wide basis.

The Properties imported with a BOM Model or that you define in either the Item Master or the Model view are called **User Properties**. Properties in the Item Master are associated with an Item Type. For more information about Item Types, see Section 3.4.1 on page 3-8.

Model nodes also have **System Properties**, which are attributes that all nodes have, but are different for each node type. When you import BOM data, the System Properties are also imported into Configurator Developer. You may want to use System Properties when building configuration rules using the Advanced Expression Editor. For more information, see Section 5.7.5 on page 5-26.

Table 3–1 lists each System Property and the node type for which each Property exists.

Table 3–1 System Properties

System Property	Node Type
Name	All nodes

Table 3–1 (Cont.) System Properties

System Property	Node Type
InstanceCount	Instantiable BOM Models, Components, References to non-BOM Models, Connectors
MinimumInstances	Instantiable BOM Models, Components, References to non-BOM Models
MaximumInstances	Instantiable BOM Models, Components, References to non-BOM Models
Quantity	BOM Models, BOM Option Classes, BOM Standard Items, Feature Options
Selection	BOM Models, BOM Option Classes, List of Options Features with a min / max **Number of Selections** of 1 / 1

For more information about instantiable components, see Section 3.5, "Multiple Instantiation in Solution Models" on page 3-14.

3.3 Effectivity

Effectivity is a feature that allows you to model a product whose structure or rules change over time. The effectivity you assign to Model structure nodes and to configuration rules determines whether the node is available or the rule is active in a runtime Oracle Configurator, based on parameters passed by the application that is hosting the Configurator window. These parameters are specified in the host application's **session initialization message** and include the date, time, and one or more Usage(s). All nodes that are not effective when the Model is invoked by the hosting application do not appear in the runtime UI. Similarly, all configuration rules that are not effective are ignored. For more information about the session initialization message, see the *Oracle Configurator Implementation Guide*.

When you import a BOM Model, the effective dates defined for each BOM item in Oracle Bills of Material are also imported, but are read-only in Configurator Developer. You cannot assign an Effectivity Set or Usages to imported BOM items.

You control effectivity by assigning a range of dates, an **Effectivity Set**, or one or more **Usages** to specific nodes in your Model and any configuration rules. (Effectivity Sets and Usages are described in the following sections.) The effectivity on a Model's root node is read-only. However, you can modify the effectivity of a *referenced* Model when viewing its parent in the Model window by changing the effectivity of the Reference node. For more information about Model referencing, see Section 2.1 on page 2-1.

You can see how effectivity affects the runtime structure of your Models and rules using the **Test** button in Configurator Developer. See Section 7.1, "The Test / Debug Module" on page 7-1.

You cannot specify effectivity for Functional Companions using Configurator Developer, but you can do this within the Functional Companion code using the `isEffective` method. For more information, see the *Oracle Configuration Interface Object (CIO) Developer's Guide.*

3.3.1 Effectivity Sets

Create Effectivity Sets to define an effective date range that can be shared by many Models, Model nodes, and configuration rules simultaneously. When you modify an Effectivity Set's date range, the change affects all Model nodes and rules to which it is assigned. Therefore, if you expect to use a specific effectivity date range for more than a very limited number of Model nodes, it is better to define and assign an Effectivity Set rather than specifying an effective date range for each node and rule within your Model.

You can assign Effectivity Sets to Components, Features, Options, rules, Totals, Resources, References, and Model publications. You can also assign an Effectivity Set to Models that you create in Configurator Developer, but not to imported BOM Models; this is because the effective dates are imported with the BOM Model and are read-only in Configurator Developer. To change to the effectivity assigned to imported data, you must edit the BOM in Oracle Bills of Material.

You can also assign an Effectivity Set to rules that are part of a Rule Sequence. See Section 5.12.2, "Rule Sequences and Effectivity Sets" on page 5-48.

An Effectivity Set can be always effective, never effective, or effective only within the range of dates that you specify.

Examples of how Effectivity Sets affect Model components include:

- Components that are not effective do not appear in the runtime UI.
- Configuration rules that are not effective are ignored (not executed) at runtime.
- A configuration rule can fail at runtime if a rule participant is not effective. For example, a Logic rule specifies that you must have a compression gauge and a feeder hose to configure an air compressor. If the feeder hose is unavailable due to its effectivity, the rule will be triggered at runtime. Oracle Configurator displays an error message when a rule fails due to one or more ineffective components.

For information about modifying Effectivity Sets, see Section 8.3, "Managing Repository Entities" on page 8-3.

3.3.2 Usages

You assign Usages to non-BOM nodes and configuration rules to control their availability at runtime. Model nodes that are not available based on the Usage provided by the hosting application at runtime do not appear in the UI. Similarly, if a rule is not assigned to the Usage provided for the current configuration session, it is ignored at runtime. Usages consist of any text string that you enter and can be defined in either the Repository or Model window.

Like Effectivity Sets, Usages provide a method of controlling the availability of Components, Features, Options, rules, Totals, Resources, and referenced Models, within a configuration model and the availability of Model publications from a host application. You can also assign Usages to Models that you create in Configurator Developer, but *not* to imported BOM Models or BOM Option Classes. However, you can assign a Usage to a node that *references* a BOM Model. You can assign Usages independently or in addition to an Effectivity Set.

When you create a new Usage, Configurator Developer automatically assigns it to all of the Models, Model nodes, and configuration rules you have defined. To make a node or rule unavailable for a specific Usage, you must modify the list of Usages for that node or rule in the Model window.

When a configuration model is invoked by a host application such as Oracle Order Management or *i*Store, the Usage(s) provided by the host application determine which parts of the Model are displayed to the end user. Any nodes that are not assigned to the Usages specified do not appear in the runtime Oracle Configurator.

For an example of how you can use Usages to limit a publication's availability to a hosting application, see the *Oracle Configurator Implementation Guide*.

Usages can be a very powerful tool you can use to manipulate the availability of Model nodes and Model publications, based on a variety of business requirements. Therefore, we recommend that you plan for and implement Usages very carefully when developing configuration models.

For information about creating and modifying Usages, see Section 8.3, "Managing Repository Entities" on page 8-3.

3.4 Imported BOM Models

Before you can import a BOM into Configurator Developer, it must be defined in Oracle Applications.

Each BOM imported from Oracle Bills of Material corresponds to one BOM Model in Oracle Configurator Developer. The name of the BOM Model corresponds to the BOM name and Model nodes mirror the imported BOM structure.

A required BOM component is also called a mandatory component. Mandatory components are not imported into Oracle Configurator Developer with the BOM, since they are not configurable. However, there is one exception to this rule: a mandatory component *will* be imported into Configurator Developer if it contains optional components, since in this case the required component is configurable. For more information about mutually exclusive and required components, see Section 3.4.2, "Imported BOM Rules" on page 3-9.

> **Note:** Do not confuse BOM components with Component nodes in Oracle Configurator Developer. BOM components are simply Oracle Inventory Items (Option Classes or Standard Items) that make up the BOM. For more information about Component nodes, see Table 4–1 on page 4-2.

Typically, an integrated Oracle Configurator in Oracle Applications is based on an existing BOM Model defined in Oracle Bills of Material. During the import process, Oracle Configurator Developer automatically populates the hierarchical Model tree and the Item Master with the BOM Model data.

The types of BOM Models that are configurable (and therefore are imported) are Assemble to Order (ATO) and Pick to Order (PTO) BOM Models. These BOM Models are defined in Oracle Bills of Material using item data defined in Oracle Inventory. The imported data is read-only in Configurator Developer because it must correspond to the BOM defined in Oracle Bills of Material throughout the business process. However, you can add additional Components, Resources, Totals, and so on to the Model to meet your configuration requirements.

A PTO BOM Model may also be a **Container Model**. This type of Model allows its configurable components to be updated in a runtime Oracle Configurator. For more information, see Telecommunications Services Ordering documentation.

3.4.1 Item Types and Imported BOM Properties

When you import a BOM Model from Oracle Bills of Material, **Item Catalog Groups** defined in Oracle Inventory become Item Types in Configurator Developer. An Item Catalog Group is a related set of **Descriptive Elements** that are assigned to Inventory Items to provide additional information about an Item. Examples of Descriptive Elements include color, length, style, and weight. The Descriptive Elements and **Descriptive Element Values** defined in an Item Catalog become the BOM item's Properties and Property Values, respectively (see Table 3–2 on page 3-8).

If no Item Catalog Groups are defined, the imported BOM items have an Item Type of Default Type. To view an Item's **Item Type**, go to the Model module, select an Item in the Item Master view, then expand the **Type/Properties** region.

Table 3–2 Item Data Imported from Oracle Inventory

Oracle Inventory	Oracle Configurator Developer
Item Catalog Group Name	Item Type
Descriptive Element	Item Property
Descriptive Element Value	Item Property Value

For more information about Item Catalog Groups and Descriptive Elements, see the *Oracle Inventory User's Guide*.

Configurator Developer does not allow you to delete or modify imported Items, Item Types, or Properties. For example, you cannot change the name or description of an imported Item or Item Type.

You can add Properties to imported Items or Item Types. However, if you add an *imported* Property, Configurator Developer does not allow you to then delete or modify the Property (you can modify or delete the Property if it was created in Configurator Developer). Note that you cannot export any Properties that you add to an imported Item Type to Oracle Bills of Material (that is, to update the BOM). For more information, see Section 4.5, "Modifying the Item Master" on page 4-14.

The following information about BOM items is imported into Configurator Developer:

- **Name**: The name of the item.
- **Description**: A brief description of the item.

- **Definition**: A basic definition of the item, including its Item Type, Minimum and Maximum Quantity, Default Quantity, and whether:
  - Optional children are mutually exclusive
  - The item is required in the configuration when its parent is selected
  - A decimal quantity is allowed at runtime when ordering the item (see Section 3.4.3.3, "Decimal Quantities" on page 3-11)
- **Properties**: Item attributes defined in Oracle Inventory (see Table 3–2 on page 3-8).
- **Property Values**: Item attribute values defined in Oracle Inventory.
- **Effective date**: The range of dates (inclusive) in which the item can be added to a configuration.

3.4.2 Imported BOM Rules

Basic rules that are inherent in the BOM Model are imported and automatically applied in Oracle Configurator Developer. These rules include settings that indicate whether components in the BOM are optional, mutually exclusive, and any **Quantity Cascade** calculations (see Section 3.4.5, "Quantity Cascade Calculations" on page 3-12).

- **Required** rules apply to child nodes that become required in the configuration when the node's parent is selected. At runtime, when the parent is selected Oracle Configurator automatically selects the required children.
- **Mutually Exclusive** rules apply to parent nodes from which you can choose only one out of all optional child nodes. In other words, of all non-required children, only one can be selected.

You can create and associate additional rules for all BOM items in the Model, including Resources and Totals.

Each BOM Model and BOM Option Class corresponds to a UI screen in a runtime Oracle Configurator. The UI displays BOM items in the same order that they appear in the Model hierarchy in Configurator Developer.

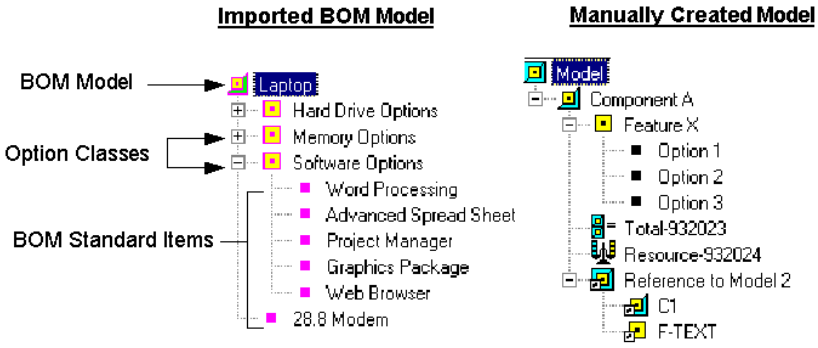
In Configurator Developer, you can view how these rules are defined for a specific BOM node by selecting it and expanding the **Definition** attribute. See Section 9.14.4, "Definition" on page 9-28.

3.4.3 Viewing Imported BOM Model Nodes

You can easily differentiate between BOM Model structure nodes and nodes that you create in Configurator Developer by the colors of the node icons. A BOM Model includes a root BOM Model node, BOM Option Classes, and BOM Standard Items. These nodes correspond to entity types of the same name in Oracle Bills of Material.

In Figure 3–1, **Laptop** is a BOM Model and is equivalent to Component A shown in the manually-created Model. **Software Options** is a BOM Option Class and is equivalent to Feature X. **Word Processing**, **Advanced Spreadsheet**, **Project Manager**, and so on are BOM Standard Items and are equivalent to Feature Options shown in the non-BOM Model.

Figure 3–1 An Imported BOM Model and a Model Created In Configurator Developer



Although BOM Model nodes are similar in many ways to nodes created in Configurator Developer, they also have some unique characteristics. For example, the initial logic state of a required item in a BOM Model is Unknown while the initial logic state for a required Feature node in a Configurator Developer Model is Logic True (a Feature is required if its **Minimum** and **Maximum Number of Selections** are both set to 1). Similarly, if the BOM Model node is set to either Logic False or User False during a configuration session, its parent node changes to the same logic state. (For more information about logic states, see Section 5.1.2 on page 5-2.)

For BOM Models, BOM Option Classes, and BOM Standard Items, the information in the **Name**, **Description**, and **Definition** attributes are gray (read-only), which indicates that their values are imported from the BOM and cannot be changed in Configurator Developer.

3.4.3.1 Properties and Imported BOM Model Structure

You can add Properties to a BOM Model, BOM Option Class, or BOM Standard Item in Configurator Developer. However, you cannot add Properties to a node that is a Reference to a BOM Model or to any of the nodes within the referenced structure (in this case, the **Add** button within the Properties attribute is disabled). You can also add Properties to Item Types associated with a node.

For more information, see Section 3.2, "Properties" on page 3-2.

3.4.3.2 Imported BOM Model Names in the Model Window

The BOM Model name that appears in the title bar of the Model window consists of the Model name as it appears in the Repository window, followed by the BOM Model's organization ID and Oracle Inventory Item ID.

For example:

```
Production V1 Test (204 52144)
```

In this example, Production V1 Test is the BOM Model name as defined in the Repository window, 204 is the organization ID, and 52144 is the Oracle Inventory Item ID. (Note that the organization ID and Item ID are *internal* values and are therefore not visible in Oracle Inventory.) In the Model window you can modify the name of a Reference to a BOM Model, but you cannot modify a BOM Model's name or description in Configurator Developer.

3.4.3.3 Decimal Quantities

In Configurator Developer, the **Decimal Quantity Allowed** box indicates whether the end user can enter a decimal quantity in a Configurator window for the selected BOM Standard Item. This read-only box appears in the **Definition** attribute in the Model window and is automatically checked for any imported BOM Standard Item that accepts decimal quantities at runtime. An end user can enter up to 9 digits after the decimal character for items that accept decimal quantities. See Section 9.14.4, "Definition" on page 9-28.

For information about importing BOM Models that use decimal quantities and configuring items with decimal quantities at runtime, see the *Oracle Configurator Implementation Guide*. For information about using BOM Standard Items that allow decimal quantities in a Numeric rule with an advanced expression, see Section 5.13.2.4, "Functions" on page 5-59.

When adding non-BOM nodes to an imported BOM Model (such as Features to generate guided buying or selling questions), you can specify a data type of either

Integer Number or Decimal Number. This is true regardless of whether you import BOM Standard Items as decimals or integers. Totals and Resources display up to two values after a decimal point (for example, .12) while numeric Features display up to nine values after a decimal (for example, .123456789).

Note: Oracle Bills of Material allows you to create BOM Models in which a divisible (decimal) parent has one or more indivisible (integer) children. However, Oracle Order Management does not allow users to order a BOM defined this way. Additionally, Oracle Configurator Developer cannot generate logic for such a Model. For example, an Option Class that allows decimal quantities must not contain Options that are defined as integers.

3.4.4 Extending a BOM Model in Configurator Developer

You can include additional aspects of your configuration problem by adding structure (non-BOM nodes) to extend an imported BOM Model. For example you might want to create Totals and Resources to keep track of a quantity or add nodes to present **guided buying and selling** questions to Oracle Configurator end users. You can add non-BOM structure to a BOM Model, but not to BOM Option Classes or BOM Standard Items.

Note: If you want non-BOM nodes that you add to a BOM Model to appear at runtime, you must generate a DHTML (Components Tree) UI; Model nodes that you create in Configurator Developer do not appear in a Java applet (BOM Model Tree) User Interface.

See the *Oracle Configurator Implementation Guide* for more information about importing data into Configurator Developer.

3.4.5 Quantity Cascade Calculations

Quantity Cascade calculations determine the final quantity requirements for a selected BOM item's children.

When you import a BOM Model, all of the parent-to-child relationships that exist among the components in the BOM are maintained. A parent component (such as a BOM Option Class) may have multiple children; some required, some optional, and some mutually exclusive. When a parent is selected in a runtime Oracle

Configurator, all of its required children are also selected. Similarly, when any child is selected, its parent is also selected.

Each component in a BOM is imported with a Minimum Quantity, Maximum Quantity, and Default Quantity. The Minimum Quantity is the smallest number of the selected node allowed per parent. The Maximum Quantity is the largest number of the selected node allowed per parent. The Default Quantity is how many of the selected node will be ordered (per parent) if this value is not modified in the selection process.

Whenever the number of a selected option is greater than zero, a Quantity Cascade calculation is performed which results in the actual quantity (or count) for that BOM item. The Quantity Cascade calculation is:

```
child node actual quantity = (parent node actual quantity) X (child node default quantity)
```

This calculation is true when the end user selects the parent item (for example, a BOM Option Class) and one of its children, but does not change the amount of the child item (a BOM Standard Item). Therefore, the count of the child is derived from the equation shown above.

However, if the end user enters a different amount for the child (BOM Standard Item) and then changes the amount of the parent (BOM Option Class), the Quantity Cascade calculation is:

```
( new child node amount / current child node amount ) X (old parent node amount)
= new parent node amount
```

For example, OptionClassA (the parent node) contains Option1 (the child node). Option1 has an initial count of 2. The end user sets Option1 to 4 and then changes OptionClassA to 3. The Quantity Cascade calculation determines a new amount for Option1 as follows:

```
( 4 / 2 ) X 3 = 6
```

Note that the calculation ensures that the new amount of the child node (6) is a multiple of its default quantity (2).

These Quantity Cascade relationships reflect the relationships between components that are built into the Oracle BOM to ensure that the BOM is filled properly. For example, consider the BOM for a car that specifies four wheels and five lug nuts for each wheel. If you select the car, that means you must have four wheels and twenty lug nuts. Similarly, if you select one wheel, that forces selection of the car, which

forces the Quantity Cascade calculation, which selects four wheels and 20 lug nuts. The Numeric rules you build in Configurator Developer respect these Quantity Cascade relationships. If you build a rule that contributes to the count of lug nuts, and the end user uses that rule to select 25 lug nuts, the runtime Oracle Configurator:

- Determines the minimum number of cars for the specified number of lug nuts (2 cars)
- Calculates the number of wheels required (8) and the number of lug nuts needed (40)

All BOM Standard Item actual quantities are calculated this way and are propagated from the root BOM node down through the entire BOM Model.

Initialization Behavior

There are some special considerations for handling BOM quantity values. See the *Oracle Configurator Implementation Guide* for a description of the model_quantity initialization parameter.

3.5 Multiple Instantiation in Solution Models

Multiple instantiation is the ability to create and individually configure multiple occurrences of a Model or Component in a runtime Oracle Configurator. A configuration model that contains Models or Components that can be instantiated multiple times is called a **Solution Model**.

At runtime, the end user accesses an instance of a configuration model, as well as an instance of each component contained within the Model. (This section uses "components" when referring to Model and Component instances at runtime.) The end user configures component instances separately by selecting from available options.

For example, a computer system can be represented by a Solution Model. A computer may contain a number of different servers, printers, and personal computer (PC) workstations. Each PC workstation in the system represents one instance of a configuration model, and each instance of the PC workstation can be configured differently.

To continue the example, one instance of the PC workstation can be configured with a 21 inch flat screen monitor, 10GB of disk space and 512 KB RAM, whereas another instance of the PC workstation has a 17 inch screen, ergonomic keyboard, 256 KB

RAM and 4 GB of disk space. These two workstations are part of the computer system Solution Model.

3.5.1 Multiple Instantiation Conditions

Not all Model and Component nodes can have multiple instances. However, nodes that can have multiple instances and those that cannot are not different when it comes to publishing, batch validation, and saving and restoring configurations.

3.5.1.1 Hosting Applications

The hosting application must pass a parameter in the session initialization message to indicate that it supports multiple instantiation. See the *Oracle Configurator Implementation Guide* for information about the sbm_flag initialization message parameter.

In Oracle Applications, each instance appears as an item under its parent on a separate line in the quote or order.

3.5.1.2 Data Import

There are no additional settings in Oracle Inventory that are required to enable Items to be instantiated multiple times in a runtime Oracle Configurator.

The imported root BOM in a Solution Model must be a PTO (Pick-to-Order) BOM Model. See Section 3.5.1.4 on page 3-16 for more information.

Refreshing an imported BOM Model updates the default Quantity specified in Oracle Bills of Material, but does not update the values of the **Instance** attribute in Configurator Developer. This attribute determines the minimum and maximum number of component instances that can exist in a configuration. See Section 3.5.2 on page 3-17.

For more information about data import, see the *Oracle Configurator Implementation Guide*.

3.5.1.3 Top-Level Root

The top-level root node cannot have multiple instances at runtime.

The top-level root node of a BOM Model must be a PTO Model to support multiple instances of its components.

A Model created in Oracle Configurator Developer supports multiple instantiation of its components, with some restrictions. See Section 3.5.1.5 on page 3-16.

3.5.1.4 Nodes that Can Have Multiple Instances

Configurable components include Models, Components, and BOM Models. The **Instances** attribute appears for all configurable components, and includes two fields: **Minimum** and **Maximum**. If you can modify a configurable component's **Instances Minimum** or **Maximum** values in Configurator Developer, then the component may be instantiated multiple times at runtime.

When an ATO BOM Model is a child of another ATO Model, you can enter a value of 0 or 1 in the **Instances Minimum** field for the child ATO. However, the **Instances Maximum** field is set to 1 by default and is read-only. Therefore, if the child ATO Model's **Instances Minimum** is 0, an Oracle Configurator end user can create only *one* instance of it at runtime. Additionally, the child ATO Model that you want to be able to instantiate must have the **Optional** check box selected in Oracle Bills of Material.

For more information about the **Instances** attribute, see Section 3.5.2 on page 3-17.

The following nodes can be instantiated *multiple* times:

- A PTO BOM Model that is a child of a PTO BOM Model
- An ATO BOM Model that is a child of a PTO BOM Model
- A Reference to a Model created in Configurator Developer (a Configurator Developer Model)
- Components of a Configurator Developer Model

> **Note:** The **BOM Item Type** field in Configurator Developer indicates whether the selected BOM item is an ATO Model, PTO Model, Option Class, or Standard Item.

3.5.1.5 Nodes that Cannot Have Multiple Instances

The following nodes cannot have multiple instances at runtime:

- The top-level (root) Model node
- A PTO BOM Model that is a child of a Configurator Developer Model
- BOM Option Classes, BOM Standard Items, Features, Options, Totals, Resources, Connectors
- A child of an ATO BOM Model (unless the child is a Configurator Developer Model and the Model does not have BOM node descendants)

An ATO BOM Model that is a child of another ATO BOM Model can be instantiated only *once* at runtime (see Section 3.5.1.4 on page 3-16).

- A Model that is a child of an BOM Option Class
- Any Model invoked by a host application that does not support Solution Models (see Section 3.5.1.1, "Hosting Applications" on page 3-15)

3.5.2 Changing the Minimum and Maximum Number of Instances

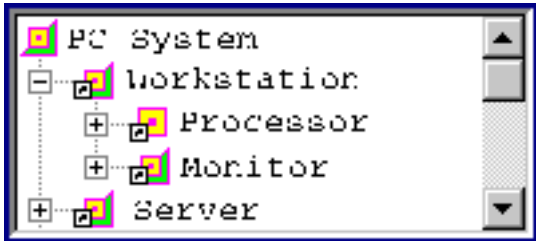
There are two ways to change the minimum and maximum number of component instances allowed at runtime: by modifying the node's minimum and maximum number of Instances, and defining a Numeric rule that changes the number of instances allowed at runtime based on end user selections.

3.5.2.1 Modifying the Instances Attribute in Configurator Developer

In Configurator Developer, modify a node's **Instances** attribute if you want end users to be able to create multiple instances of that component at runtime. (For a list of nodes that support multiple instantiation, see Section 3.5.1.4 on page 3-16.) You can edit the Minimum and Maximum **Instances** on a Reference to an ATO or PTO BOM Model only if the Reference is a direct child of a PTO BOM Model. To edit the **Instance** count on a nested Reference (that is, a Reference within a Reference), you must open its parent Model in the Model window for editing.

For example, Figure 3–2 on page 3-17 shows a PC System, a PTO BOM Model, with References to a Workstation, which is a PTO BOM Model, and a Server, which is an ATO BOM Model. Additionally, the Workstation references Monitor, which is also a PTO BOM Model. You can edit the minimum and maximum **Instances** for the Workstation and Server while editing the PC System Model. However, to change the number of **Instances** for the Monitor Model, you must first open its parent Model (Workstation) for editing, then select the Monitor Reference node.

Figure 3–2 Setting the Instance Attribute on Referenced Nodes



Note: Changing the **Instances Minimum** value for a component within a published Model may change the number of instances that exist in a saved configuration. For example, decreasing the **Instances Minimum** in Configurator Developer may cause some instances of the component to be lost when the configuration is restored (Oracle Configurator displays a message indicating that a validation failure occurred in this case). Similarly, increasing the **Instances Minimum** value may create additional instances in a restored configuration.

3.5.2.2 Using Numeric Rules

You can dynamically increase or decrease the number of instances allowed at runtime by defining a Numeric rule. For more information, see Section 5.7.5, "Building Rules Using Node Properties" on page 5-26.

3.5.3 Loading Solution Models

At runtime, an Oracle Configurator window loads one instance of the selected Solution Model and the minimum number of **Instances** specified for each component in Oracle Configurator Developer.

If a component's minimum number of **Instances** is zero, then only the component name and an **Add** button appear in the component's parent screen; no instances of that component are initially loaded. The end user clicks the **Add** button to add new instances of the component to the configuration. See Section A.2, "Creating Instances at Runtime" on page A-6.

Components that have a minimum number of **Instances** of one or more exist when the configuration session begins. If the minimum and maximum number of **Instances** are not equal, the end user can create new instances of the component by clicking the **Add** button, until the maximum number of instances is reached. See Section 6.1.3.3, "Add and Delete Buttons" on page 6-8.

Note: Instantiable components with a large number of minimum **Instances** may increase the time required to initially load the Model in an Oracle Configurator window.

3.5.4 Multiple Instantiation Modelling Guidelines

It must be possible for an end user to add all necessary components to a configuration in the absence of any previous end user requests or default option selections. In other words, it is important that you design your Model and rules such that components can be added (and deleted) regardless of whether a Defaults rule or the end user has selected any options.

Therefore, in a multi-component system, it is recommended that you use validation logic instead of rules that cause contradictions. Otherwise, Oracle Configurator end users might not be able to create a valid configuration.

See Example 3–1 on page 3-19 for more information.

Example 3–1 Adding a Component Results in a Contradiction

Your Model contains a List of Options Feature, called Feature Z. This Feature is a child of Component X, whose **Minimum** and **Maximum Number of Selections** are both set to 1. Your Model also contains the following rules:

```
Rule 1: Option A contributes 5 to Total Watts
Rule 2: Total Watts < 0 Excludes Always True
Rule 3: Feature Z Consumes 1 from Total Watts
```

At runtime, the end user selects Option A, so Total Watts becomes 5. The end user then clicks an **Add** button to add another instance of Component X (and by extension, Option Feature Z) to the configuration. At this point, a contradiction message appears because the value of Total Watts is less than zero. This is because when the end user adds Component X, the runtime Oracle Configurator:

1. Retracts the user's previous request (in this example, this means Option A is deselected, which sets Total Watts back to its initial value of zero)
2. Adds the component to the configuration

Because Rule 2 raises a contradiction, Option A must be deselected before the end user can proceed.

Note: The runtime Oracle Configurator also retracts previous requests and default selections whenever an end user connects or disconnects components. Connectivity is explained in Section 3.6 on page 3-20.

A better approach is to construct a rule that uses *validation* logic, so the user can acknowledge the warning and then proceed with the configuration. You can do this in the Model above by defining two rules that use Watts Feature, which is an Integer Feature that has both an initial value and a **Minimum** of 0 (zero).

You define the following rules:

```
Rule 1: Option A contributes 5 to Watts Feature
Rule 2: NotTrue(Component X) Consumes 1 from Watts Feature
```

Now, when the end user adds Component X and violates Rule 2, Oracle Configurator displays a warning, but does not require the end user to make changes before proceeding. Therefore, the end user successfully adds the component and can complete the configuration. (This assumes that the other rules listed in the first part of the example do not exist, or are disabled.)

Another example of defining rules to use validation logic is provided in Section 5.9 on page 5-29.

3.6 Connectivity and Networks

In some Models, the Oracle Configurator end user not only makes selections from the available components, but must also specify how some or all of the components are *connected* before the configuration is valid and complete. For example, a piece of office furniture consists of a desktop, several shelves, and a file cabinet. The pieces can be connected to form a single unit, but multiple configurations are possible. Although customers indicate how they want the unit to be assembled, the end result must be something that the factory can produce. A product such as this requires *connectivity* to be defined between specific Model components and rules must exist to ensure that each connection, and ultimately the final assembly, is valid.

Another example of a Model that requires components to be connected is a Model that enables an Oracle Configurator end user to define a **network.** A network is a system comprised of interrelated or interconnected elements, such as telephones, computers, television stations, or even health care practitioners that are affiliated with a specific insurance provider. Each kind of network is unique, but all networks are comprised of individual components that are either literally or figuratively connected in some way.

In Configurator Developer, you build a Model that allows components to be connected at runtime by creating **Connectors**. Connectors define connectivity from a Model or Component node to any other Model in the CZ schema. If the Model chosen as the Connector's target exists at runtime, and creating the connection does

not violate any configuration rules you have defined, an end user can connect the components simply by clicking a **Choose Connection** button. (**Choose Connection** buttons are described in Section 3.6.3, "Connectors and the Runtime User Interface" on page 3-27.)

As with any Model, Models that allow connections may have constraints defined in Configurator Developer to ensure that all required connections are made, each connection is valid, and all configuration requirements are satisfied.

Example 3–2 Telephone Network Model

In a runtime Oracle Configurator, an end user configures a telephone network by defining Circuits. The user creates a Circuit by connecting two Ports. One or more configurable Ports exist per Circuit Node, and Ports include the parameter Speed. The Speed parameter includes the Options High, Medium, and Low, and any two connected Ports must have the same speed. The end user connects two Ports at a time, while constraints defined in Configurator Developer ensure that the Speed of the connected Ports is the same. When all required selections and connections are made and the configuration is valid, the data is passed back to the hosting application for downstream processing.

Note: The ability to connect Model components in a runtime Oracle Configurator is available only in a DHTML User Interface; an end user cannot connect components in a Java applet UI.

3.6.1 Connectors and Target Models

In Configurator Developer, you can create a Connector when either a Model or a Component node is selected. However, a Connector's **target** is always a Model. In other words, an Oracle Configurator end user can create a connection from:

- A BOM Model to another Model
- A Component to a Model

Before an Oracle Configurator end user can connect two components at runtime, an instance of each component must exist in the configuration. (Remember that Model and Component nodes appear as individual UI screens in a runtime Oracle Configurator. For this reason, this document uses the word "component" when referring to a UI screen generated from either node type.) This is an important consideration when building your Model in Configurator Developer, because all components that you want an end user to be able to connect at runtime must be part of the Model's structure.

In Configurator Developer, the Model that is the target of a Connector must be referenced from the Model being configured at runtime. This is because an instance of the target Model must exist in the configuration before an end user can connect the Connector's parent and the target Model (and the only way to include a Model within another Model is to create a Reference).

If an instance of the target Model does not exist at runtime, an error message appears when an end user attempts to create a connection. See Section 3.6.3, "Connectors and the Runtime User Interface" on page 3-27.

3.6.1.1 Connector Targets and Model Structure

When you create a Connector and specify a target Model, the structure of the target Model is visible beneath the Connector node and you can use nodes from the target Model when defining configuration rules for the Model containing the Connector. The target Model's rules are also visible when you are working in the Configuration Rules module. However, you cannot modify a target Model's structure or rules when working in the Model that contains the Connector (these nodes are read-only). To modify a target Model's structure or rules, you must open it for editing in the Model window. To see how the structure of a target Model appears in Configurator Developer, see Figure 3–3 on page 3-24.

Although the target Model and its structure are visible beneath the Connector node in Configurator Developer, a new instance of the target Model is *not* automatically created when another instance of the Connector's parent is added at runtime. If the configuration requires another instance of the target Model, the end user must add it by clicking the **Add** button that appears on the Connector's parent UI screen (see Figure A–1 on page A-7). For more information about adding component instances at runtime, see Section 3.5, "Multiple Instantiation in Solution Models" on page 3-14.

3.6.2 Connectors and Configuration Rules

A unique characteristic of Models that contain Connectors is that you can define configuration rules relating instances that may be connected at runtime. However, these rules do not propagate or cause contradictions until the end user actually connects the components that are participants in the rule. Then, the rule verifies that the connection is valid and propagates to other parts of the Model (depending on its definition). If the connection is valid, the Oracle Configurator end user can continue; otherwise, a contradiction message explains why the connection is invalid and the action is cancelled. For an example of this behavior, see Section 3.6.2.2, "Runtime Behavior of Rules Relating Connected Components" on page 3-26.

Before you can create rules to relate components that may be connected at runtime, you must create Connector nodes in Configurator Developer and specify the Model that serves as the Connector's target. (See Section 4.3.5, "Creating a Connector" on page 4-6.) You can then use any nodes within the structure of the Connector's target Model to define any type of configuration rule. However, Connector nodes *themselves* cannot participate in configuration rules because they do not have attributes such as a minimum or maximum number of instances, any Properties, or a logic state in the runtime UI.

Note: You *can* use a Connector node when defining a Functional Companion. See Section 3.6.2.3, "Connection Filter Functional Companion" on page 3-26.

The ability to use nodes within the structure of a Connector's target enables you to create configuration rules between **optional instantiable components**, which are Model or Component nodes that may exist at runtime, but are not required to create a valid configuration (for example, a Component or Model Reference that has a minimum number of **Instances** set to 0). For more information, see Section 5.15, "Rules that Relate Components and Models" on page 5-68.

Note: Use caution when using nodes within a target Model as participants in the parent Model's configuration rules. If a node in the target Model is modified or deleted, the rule in the parent Model becomes invalid and Configurator Developer displays an error message when you generate the **Active Model** for the parent.

Note: It must be possible for Oracle Configurator end users to connect all necessary components in a configuration in the absence of any previous end user requests or default option selections. For details, and an important model design suggestion, see Section 3.5.4 on page 3-19.

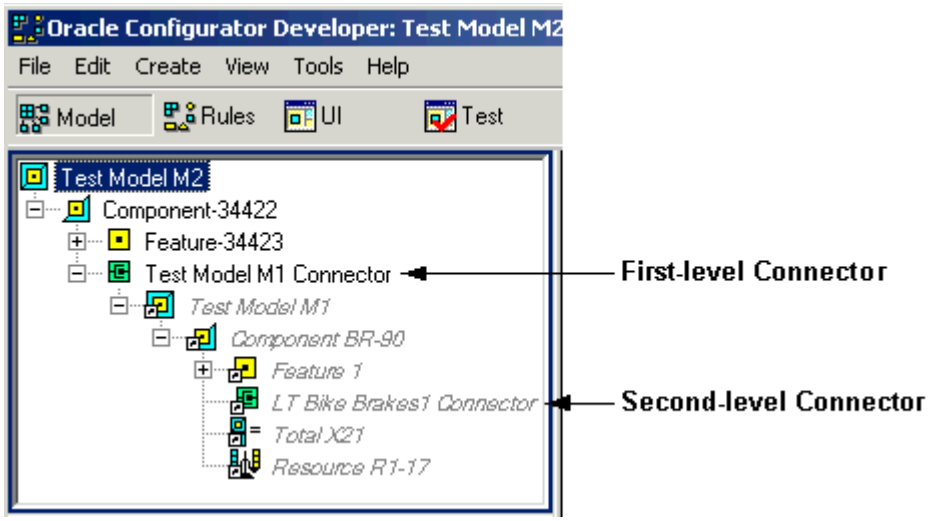
3.6.2.1 Second-Level Connectors

Oracle Configurator Developer displays the structure of a Connector's target Model so you can use nodes from the target Model's structure to build configuration rules for the Model that is open for editing. However, a target Model may also contain a Connector, and in this case the structure of the second-level (nested) Connector's

target Model is not visible in Configurator Developer. Because the second-level Connector's target Model is not visible, nodes from that Model cannot participate in the root Model's configuration rules.

Figure 3–3 shows how a Model containing both first and second-level Connectors appears in Configurator Developer.

Figure 3–3 First and Second-Level Connectors

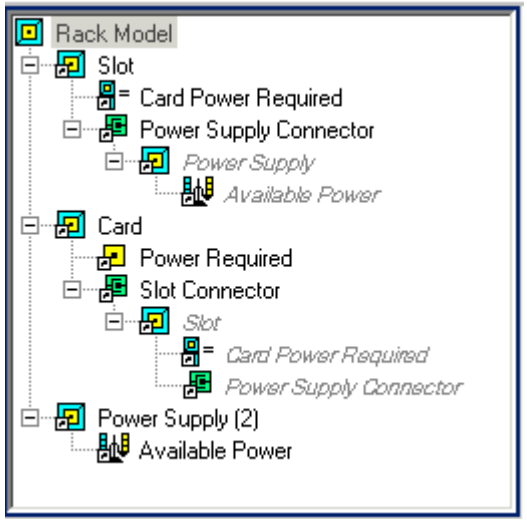


To work around the restriction of creating rules using nodes from a second-level Connector's target Model, perform the following:

1. Create an intermediate node in the first-level target Model.
2. Define a rule in the first-level target that includes the intermediate node and the node(s) you want to use in the nested Connector's target.
3. Define a rule in the root Model using the intermediate node to produce the desired results.

Example: The Rack Model shown in Figure 3–4 contains References to three Models: Slot, Card, and Power Supply.

Figure 3–4 Rack Model Structure



In the Rack Model, Card has a Connector to Slot, which in turn has a Connector to Power Supply. You need to define a rule that states Power Required consumes from the Available Power Resource in Slot. In the Rack Model, however, Power Supply is the target of a second-level Connector (in Card), so it is not possible to drag a node from Power Supply to create the rule.

To work around this restriction, define an intermediate node in Slot (in this example, the Card Power Required Total) to reflect information from Card, and then define the rule in two pieces.

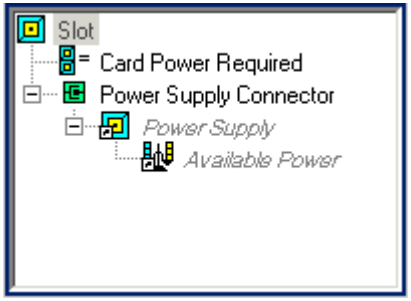
First, relate the Power Required Total in Card to the desired node in Power Supply:

```
Card.Power Required Consumes from Power Supply.AvailablePower
```

Then, create a rule in Slot relating Power Required with the intermediate node:

```
Power Required Contributes to Slot.Card Power Required
```

Figure 3–5 Slot Model Structure



3.6.2.2 Runtime Behavior of Rules Relating Connected Components

If a rule contains one or more nodes from one or more Connector's target Models, the rule does not propagate or cause contradictions until all Connectors are connected at runtime.

For example, you want to contribute a constant value to a Total whenever Option1 from Component A or Option2 from target Model B is added to the configuration at runtime. You define a Contributes to relation (Numeric rule) and include Option1 and Option2 as participants. This rule will not begin contributing to the Total, even if Option1 and Option2 are selected, until the end user connects an instance of Component A and an instance of Model B.

3.6.2.3 Connection Filter Functional Companion

You may want to control which instances of the target Model appear when an Oracle Configurator user clicks a **Choose Connection** button. To do this, define a **Connection Filter Functional Companion** and assign it to the Connector node whose target instances you want to filter.

For example, you want to prevent end users from creating more than one connection to an instance of Model X. You can define a Functional Companion that makes Model X unavailable once the end user creates a connection from another component to it. Then, assign the Connector that has an instance of Model X as its target to the Functional Companion in Configurator Developer.

For more information about creating Functional Companions, see the *Oracle Configuration Interface Object (CIO) Developer's Guide*.

For more information about creating connections at runtime, see Section 3.6.3, "Connectors and the Runtime User Interface" on page 3-27.

Note: To assign a Connector to a Functional Companion, the Connector must be defined in the Model that is open for editing (that is, not in the Connector's target Model). For example, you are editing Model A, which has a Connector to Model B. Model B also contains a Connector that is visible in the Model window. However, you cannot drag and drop the Connector in Model B when defining a Functional Companion for Model A.

3.6.3 Connectors and the Runtime User Interface

When you generate a new User Interface, Configurator Developer creates a **Connector** UI control and a **Choose Connection** button for each Connector in your Model. At runtime, these controls appear in the Connector's parent UI screen. When the end user clicks a **Choose Connection** button, a secondary window displays each instance of the Connector's target Model. The end user selects an instance and clicks **Submit** to create the connection, or **Cancel** to close the window and cancel the operation.

For more information about these UI controls, see:

- Section 6.3.4.8, "Customizing a Connector UI Control" on page 6-35
- Section 6.3.4.9, "Customizing a Choose Connection Button" on page 6-37

At least one instance of a Model that is the target of a Connector must exist in the configuration before an Oracle Configurator user can connect another component to it. If no instances of a target Model are available, a message appears when the end user clicks the **Choose Connection** button.

For example, the minimum number of instances defined for a reference to Model A is 0 (zero). Model A is the target of a Connector that is defined in Model B. When the configuration session begins, the end user cannot connect an instance of Model B to an instance of Model A because no instances of Model A exist. Therefore, until Model A is **instantiated**, no targets are available when the end user clicks the **Choose Connection** button in Model B's UI screen.

You can optionally use a Functional Companion to control whether instances of a Model appear when a user clicks a **Choose Connection** button. See Section 3.6.2.3, "Connection Filter Functional Companion" on page 3-26.

3.6.3.1 Connections and Runtime Navigation

When an end user connects two components at runtime, the name of the selected Model instance appears as a hypertext link next to the **Choose Connection** button. Clicking this link navigates to the connected Model's UI screen.

After generating a User Interface, you can optionally add a **Connections Listbox** to the UI screen of any Model that may be the target of a Connector. A Connections Listbox displays all components connected to the selected Model instance, and each component name appears as a hypertext link. Clicking a link navigates to the Connector's parent UI screen. See Section 6.3.4.5, "Adding a Connections Listbox to a Screen" on page 6-30.

3.6.4 Overview of Creating a Model that Requires Connectivity

This section assumes that you have imported a BOM Model or begun to prototype Model structure in Oracle Configurator Developer.

To create a Model that allows components to be connected at runtime:

1. Open a Model for editing in the Model window.
2. Create **Connectors** by selecting a Component and specifying a target Model. See Section 4.3.5, "Creating a Connector" on page 4-6.
3. Use Connector target Model structure to build configuration rules that constrain elements of connected runtime components (optional). Section 5.5, "Building Configuration Rules" on page 5-20.
4. Generate a new User Interface. Section 6.2, "Generating a New User Interface" on page 6-16.
5. Customize **Choose Connection** buttons (optional). See Section 6.3.4.9, "Customizing a Choose Connection Button" on page 6-37.
6. Customize **Connector** UI controls (optional). See Section 6.3.4.8, "Customizing a Connector UI Control" on page 6-35.
7. Create **Connections Listboxes** (optional). See Section 6.3.4.5, "Adding a Connections Listbox to a Screen" on page 6-30.
8. Unit test the Model and configuration rules in the runtime User Interface. See Section 7.1, "The Test / Debug Module" on page 7-1.

4

Constructing Model Structure

In Configurator Developer, the Model structure is shown as a hierarchical "tree view" and is comprised of data that represents product elements such as Models, Components, Features, Options, BOM Models, BOM Option Classes, BOM Standard Items, Resources, and Totals. This chapter describes how to use the Item Master, Properties, Effectivity, and imported data to create Model structure and the role of this structure when you ultimately implement a configuration model.

For information and tips on designing Models for maximum runtime performance, see the *Oracle Configurator Performance Guide*.

4.1 The Model Window

You use the Model window in Configurator Developer to view and modify the structure of your configuration models. The Model window consists of three distinct "views" in which you see a Model's structure, configuration rules, or User Interface. Each of these views has a corresponding **module** from which you can modify the Model's structure, rules, and UI. See Section 9.2, "The Model Window Modules" on page 9-1.

Model structure represents product data in a hierarchical "tree" and determines how the default UI is generated when an end user opens the Model in a Configurator window. The structure contains imported structure as well as additional nodes that define guided buying or selling questions and hidden elements that support totals, resources, and rules. In the Model view, imported BOM Model structure and any referenced Models are read-only and cannot be modified.

4.2 Model Structure

Constructing the Model structure is your first task in creating a configuration model. Importing a BOM from Oracle Bills of Material creates an imported BOM Model in Configurator Developer. You can then create additional structure to extend the imported BOM Model to include additional aspects of your configuration problem. This added structure does not appear in the Java applet User Interface, but can be accessed by launching a DHTML UI. See Section 6.1.2, "How the Oracle Runtime Configurator Displays the Model" on page 6-2.

When you build a Model, you translate your knowledge about how parts are configured into the products you sell into terms that Oracle Configurator Developer, and ultimately the Oracle runtime Configurator, can use. The Model is organized into a hierarchy called a **tree**. Configurator Developer provides the following **node** types to build your Model: Models, Components, Features, Options, Resources, and Totals. BOM Models imported from Oracle Bills of Material may include other BOM Models, as well as BOM Option Classes and BOM Standard Items. See Section 3.4, "Imported BOM Models" on page 3-7.

Table 4–1 lists each node type that may be part of a Model's structure in Configurator Developer.

Table 4–1 Configurator Developer Node Types

Node Type	Description
Model	The highest level of structure. A Model can contain Components and References to other Models.
Component	A configurable part of a Model. A Component can contain Components, Features, Resources, and Totals.
Connector	Enables an end user to link two components at runtime by selecting from a list of instantiated Models. A Model can have one or more Connectors.
Feature	A Feature can be a List of Options, an Integer Number, a Decimal Number, a True or False value, or a Text value. A Feature of type List of Options can contain Options.
Option	An Option is part of a List of Options in a Feature. It is often an object represented by an Item in the Item Master.
Total	A Total keeps track of a quantity. A Total can have a positive or negative value. Use Numeric Rules to contribute to and consume from Totals.

Table 4–1 (Cont.) Configurator Developer Node Types

Node Type	Description
Resource	A Resource also keeps track of a quantity. The value of a Resource can be positive or zero, but a Resource is violated if its value becomes negative at runtime (that is, it is over-consumed). Use Numeric Rules to contribute to and consume from a Resource.
BOM Model	Equivalent to a Component in Configurator Developer. Can be either an assemble to order (ATO) or pick to order (PTO) Model created in Oracle Bills of Material and may include other BOM Models. Typically contains BOM Option Classes, which include BOM Standard Items.
BOM Option Class	A group of related BOM Standard Items. BOM Standard Items are imported with the BOM from Oracle Bills of Material, and are Equivalent to a Feature Option in Configurator Developer.
BOM Standard Item	Any Oracle Inventory Item that can be a component on a bill, including purchased items, subassemblies, and finished products. Equivalent to a Feature Option in Configurator Developer.

To view a summary of your Model's structure, rules, and Item Master, generate a Model Report. See Section 9.5 on page 9-5.

4.3 Building Model Structure

Follow the steps described in the following sections to add Model structure elements to an imported BOM Model, or to build a demo or prototype system. To test your Model, see Section 7, "Testing and Debugging" on page 7-1.

In the Configurator Developer Model window, you can create supplementary structure under a BOM Model node. This structure can include the following types of nodes:

- Components (see Section 4.3.1, "Creating a Component" on page 4-4)
- Features (see Section 4.3.2, "Creating a Feature" on page 4-4)
- Totals and Resources (see Section 4.3.4, "Creating a Total or Resource" on page 4-5)
- References to Component-based Models; in other words, Models that do not contain BOM structure (see Section 2.1.7, "Creating a Model Reference" on page 2-7)

- Connectors (see Section 4.3.5, "Creating a Connector" on page 4-6)

You can also create Model structure automatically using **Populators**. See Section 4.4 on page 4-8.

> **Note:** When creating new Model structure, enter node names that are unique, meaningful to the Oracle Configurator end user and, if possible, descriptive of their intended purpose.

4.3.1 Creating a Component

A Component is a part of a product that the end user can configure.

To create a Component:

1. Select a Model or Component node.
2. Choose **New Component** from the **Create** menu or use the right mouse button to select **New Component** from the shortcut menu.
3. Enter a name for the Component.
4. Enter the following attributes for the Component:
   - "Name" on page 8-13
   - "Description" on page 8-13
   - "Instances" on page 9-29
   - "Visibility" on page 9-28
   - "Properties" on page 9-30
   - "Populators" on page 9-31
   - "Effectivity" on page 9-31

4.3.2 Creating a Feature

A Feature is an element that can be selected at runtime when configuring a Component. For example, you might create a Feature node called Color and create Options of that Feature called Red, Green, and Blue.

To create a Feature:

1. Select a Component node.

2. Choose **New Feature** from the **Create** menu or use the right mouse button to select **New Feature** from the shortcut menu.
3. Enter a name for the Feature.
4. Enter the following attributes for the Feature:
   - "Name" on page 8-13
   - "Description" on page 8-13
   - "Type" on page 9-23
   - "Visibility" on page 9-28
   - "Properties" on page 9-30
   - "Populators" on page 9-31
   - "Effectivity" on page 9-31

4.3.3 Creating an Option

An Option is a possible enumerated value of a Feature.

To create an Option:

1. Select a Feature node.
2. Choose **New Option** from the **Create** menu or use the right mouse button to select **New Option** from the shortcut menu.
3. Enter a name for the Option.
4. Enter the following attributes for the Option:
   - "Name" on page 8-13
   - "Description" on page 8-13
   - "Visibility" on page 9-28
   - "Properties" on page 9-30
   - "Effectivity" on page 9-31

4.3.4 Creating a Total or Resource

You can create Totals and Resources as child nodes of Components. A Total acts as a numeric variable in your Model. A Resource is similar to a Total, but a Resource ensures that the quantity it is keeping track of does not fall below zero (become

over-consumed). A Total can be used as a constant, or set when the end user makes another selection. You can use Configuration Rules to contribute quantities to or consume quantities from both Totals and Resources. See Section 5.7, "Numeric Rules" on page 5-21 for details about applying Configuration Rules to Totals and Resources.

You can enter a decimal value for the **Initial Value** of a Total or Resource. However, the runtime Oracle Configurator rounds values to two decimal places for these nodes (for example, an **Initial Value** of 3.12464536 displays as 3.12 at runtime).

To create a Total or Resource:

1. Select the Component node where you want to create the Total or Resource.
2. Choose **New Total** or **New Resource** from the **Create** menu or use the right mouse button to select **New Total** or **New Resource** from the shortcut menu.
3. Enter a name for the Total or Resource.
4. Enter the following attributes for the Total or Resource:
   - "Name" on page 8-13
   - "Description" on page 8-13
   - "Initial Value" on page 9-26
   - "Visibility" on page 9-28
   - "Properties" on page 9-30
   - "Violation Message Attribute" on page 9-31 (this attribute is for Resources only)
   - "Effectivity" on page 9-31

4.3.5 Creating a Connector

A **Connector** is a node you create that enables an Oracle Configurator user to define connections between component instances at runtime. In Oracle Configurator Developer, a Model may have one or many Connectors, and you can specify whether a connection is optional or required for a valid configuration. A Connector can have only one Model as its target.

For example, you create a Connector in Model A and choose Model B as its **target**. At runtime, the Oracle Configurator user can connect an instance of Model A to an instance of Model B by clicking a **Choose Connection** button.

For general information about Connectors, see Section 3.6, "Connectivity and Networks" on page 3-20. For information about runtime behavior, see Section 3.6.3, "Connectors and the Runtime User Interface" on page 3-27.

To create a Connector:

1. In the Model module, select either a BOM Model or a Component node.
2. Click the right mouse button and choose **New Connector** from the menu, or choose **Create > New Connector**.

   When you do this, the Model window appears. This window displays all Models in the CZ schema you are logged into, including the Model in which you are creating the Connector.
3. Select a target Model, then click **OK**.

   After selecting a target, you can optionally click the **Go To** button in the Attributes view to open the target Model in the Model window for editing.

   Note: If you make a mistake and need to change the Connector's target, you must delete the Connector, recreate it, and then choose a different target Model.
4. Optionally modify the name and description of the Connector in the fields provided.
5. If connecting this Connector at runtime is not required to create a valid configuration, expand the **Definition** attribute and deselect the **Connection Required** box.
6. If you do *not* want the Connector to appear at runtime, expand the **Visibility** attribute and deselect the **Display in User Interface** box.
7. Expand the **Effectivity** attribute if you want to enter an effective date range and assign an Effectivity Set or Usages to the Connector. See Section 3.3, "Effectivity" on page 3-4.
8. Verify that the Model that is the target of the Connector will exist at runtime. If a Connector's target is not part of the Model you are editing, be sure to do one of the following:
   - Create a Reference to the Connector's target Model

- Create a Reference from the Connector's target Model to the Model you are currently editing (to do this, you must first open the target Model for editing)
- Reference both the Connector's parent Model and the Connector's target Model from the same (parent) Model

See Section 2.1.7, "Creating a Model Reference" on page 2-7.

Note: If the target Model does not exist in your Model's structure, the target Model cannot be instantiated at runtime. Therefore, no targets will be available when an end user clicks the **Choose Connection** button at runtime. See Section 3.6.1, "Connectors and Target Models" on page 3-21.

4.3.5.1 Connectors and User Interface Objects

When you create a new User Interface for a Model that contains Connectors, Configurator Developer generates the following UI objects for each Connector:

- A **Choose Connection** button (see Section 6.3.4.9, "Customizing a Choose Connection Button" on page 6-37)
- A **Connector** UI control (see Section 6.3.4.8, "Customizing a Connector UI Control" on page 6-35)

4.4 Using Populators

Use **Populators** to automatically create Products, Components, Features, and Options within a Model using Items, Item Types, and Properties that exist in the Item Master. When you run a Populator, it creates the new nodes as children of the node on which it was defined. For example, if you define a Populator on a Component node, it creates new Components or Features as children of that node. If you use Populators to build Model structure from Items, any Properties and Property values associated with the Items are also incorporated into the Model. Properties and their values are *not* incorporated into the Model when you use Item Types to build Model structure.

The primary benefit of using Populators is that the system maintains a permanent link from the nodes a Populator creates to the data in the Item Master. Therefore, when data in the Item Master changes, you can update all nodes created using Populators simply by "re-populating" the Model. See Section 4.4.3, "Repopulating Model Data" on page 4-13.

You can also delete any Model structure created using a Populator by deleting the Populator. See Section 4.4.4, "Deleting a Populator" on page 4-14. If you cut and paste a node that has one or more Populators, the copied nodes also have the same Populators as the original node. However, when you *copy* a node, the Populators are not assigned to the new node.

Although you can also drag and drop Items and Item Types from the Item Master to create Model structure, nodes created using this method have no connection to the data in the Item Master and cannot be easily updated when the data changes.

Populators are *not* typically used with imported data, because using imported data to create Model structure can create maintenance problems when you refresh the Model.

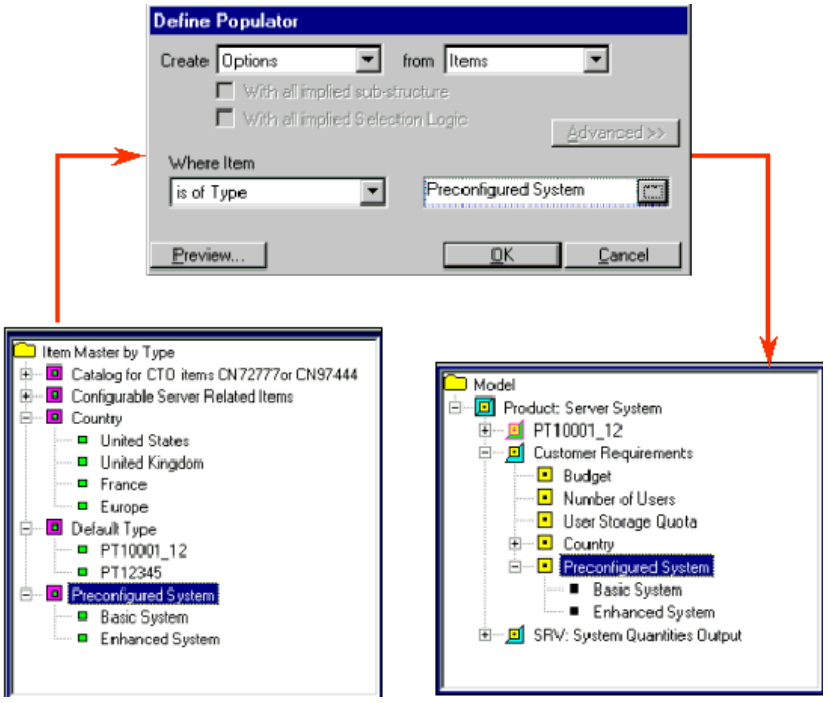
Note: You can repopulate one or more Models without logging into Configurator Developer by running an Oracle Applications concurrent program. For more information, see the *Oracle Configurator Implementation Guide*.

4.4.1 The Define Populator Dialog

Use the Define Populator dialog box to create a Populator or edit an existing Populator. The **Preview** button in the lower left of the dialog box shows you the data that the Populator selects from the Item Master. Use this preview to verify that the Populator is selecting the data you want before adding it to your Model.

Figure 4–1 on page 4-10 provides an example of how you might define a Populator in Configurator Developer.

Figure 4–1 Defining a Populator



The Name and Description Fields

Use these fields to enter a unique name and description for the Populator. You must enter a unique name when defining a new Populator. Since you can associate multiple Populators with a single node, a meaningful description can help others determine a Populator's function.

The Create Field

Use the **Create** field to specify the type of nodes the Populator creates. The type of nodes you can create depends on the node on which you are defining the Populator. Table 4–2 summarizes the available choices.

Table 4–2 Populators and Model Nodes

If the selected Model node is a . . .	You can create . . .
Component	Components or Features

Table 4–2 (Cont.) Populators and Model Nodes

If the selected Model node is a . . .	You can create . . .
Feature	Options

The Of Type Field

If you want the Populator to create Features, use this field to specify the default Data Type of each Feature. You can modify the Data Type of individual Features after running the Populator, if necessary. For a description of each Data Type, see Section 9.14.3, "Type" on page 9-23.

If you want the Populator to create Components, the **of type** field does not appear in the Define Populator dialog.

The From Field

Use the **from** field to specify the type of Item Master data the Populator uses to create the new nodes. All types of Item Master data are available in this list, regardless of the type of node you want the Populator to create. They include:

- Item Types
- Items
- Property
- Property Values

The Where fields:

Use the remaining fields in the dialog box to specify exactly what Item Master data the Populator uses. The **Where** field is case-sensitive, so it is a good idea to use the list of values to make a selection. If you do not enter the name of the item exactly as it appears in the Item Master, Configurator Developer displays an error message.

The selection criteria you can specify are determined by the type of Item Master data you entered in the **from** field, and the Where field prompt changes based on this information. When you choose Item Types, Items, or Properties, the Where field name changes dynamically to **Where Item Type**, **Where Item**, or **Where Property**, respectively. Enter criteria in the field next to the Where field by either typing text into the field or selecting from the list of available options.

Note that you can also create *all* nodes available in the Item Master under the selected node by entering the wildcard character (%). If you do this and are not

satisfied with the results, you can remove the new structure by deleting the Populator after it runs. See Section 4.4.4, "Deleting a Populator" on page 4-14.

If you select Property Values in the **from** field, you must specify criteria differently. In this case, the Define Populator dialog box displays two fields. The first is labeled **Where Item Type is**. Select the button to the right of the field to select from a list of Item Types. The second field is labeled **And Property is**. Select the button to the right of the field to select from a list of Properties.

Table 4–3 lists the available Item Master data types, the defining criteria for the Populator, and how you enter criteria based on the selected data type.

Table 4–3 Defining a Populator

Type of Item Master Data	Available Criteria	How you Specify Criteria
Item Type	begins with ends with contains matches	Type text into the field. Item Types are selected depending on whether the Item Type name begins with, ends with, contains, or matches the text you enter.
Item	is of Type	Click button to the right of the field, then select from list of Item Types.
	begins with ends with contains matches	Type text into the field. Item Types are selected depending on whether the Item Type name begins with, ends with, contains, or matches the text you enter.
Property	is Property of	Click button to the right of the field, then select from list of Item Types.
	begins with ends with contains matches	Type text into the field. Item Types are selected depending on whether the Item Type name begins with, ends with, contains, or matches the text you enter.
Property Value	Where Item Type is	Click button to the right of the field, then select from list of Item Types.
	And Property is	Click button to the right of the field, then select from list of the selected Item's Properties.

4.4.2 Creating and Modifying Populators

Follow these steps to add a new Populator or edit an existing Populator.

1. In the Model module, select the Component or Feature node you want to populate.
2. Expand the Populators attribute. This section lists existing Populators and provides buttons to **Add** a new Populator, or **Edit** or **Delete** any existing Populators.
3. Click **Add** to create a new Populator, or select a Populator from the list and click **Edit** to modify its definition. When you click **Add** or **Edit**, the Define Populator dialog opens. For more information about each field in this dialog, see Section 4.4.1, "The Define Populator Dialog" on page 4-9.
4. Enter a unique name and description for the Populator.
5. Select the type of node you want to create from the **Create** dropdown list.
6. Select the type of Item Master data the Populator should use from the **from** dropdown list.
7. Specify selection criteria for the Populator. Choices available to you and the format for your selection depends on the selection you made in the **from** list.
8. Click **Preview** to view the effects of running the Populator before actually adding to the structure.
9. Click **Close** to close the Populator Preview dialog.
10. If necessary, modify the Populator selection criteria. Otherwise, click **OK** to run the Populator.

4.4.3 Repopulating Model Data

Repopulate Models periodically to update the Model with any data changes made in the Item Master. Repopulating automatically runs all Populators defined within the Model that is open for editing.

To repopulate a Model:

1. Open the Model in the Model window.
2. Choose **Tools > RePopulate**.

4.4.4 Deleting a Populator

Deleting a Populator removes all Model structure that the Populator originally created, as well as any child nodes. To recreate the Model structure that was deleted, you must re-create and then re-run the Populator.

To delete a Populator:

1. In the Model module, select the Product, Component, or Feature node associated with the Populator you want to delete.
2. Expand the Populators attribute.
3. Click **Delete**, then click **Ok** to acknowledge the message.

4.5 Modifying the Item Master

This section explains how to add, edit, or delete Items, Item Types, and Properties that are associated with Item Types.

For a general description of the Item Master, see Section 3.1, "The Item Master" on page 3-1.

4.5.1 Creating a New Item Type

To create a new Item Type:

1. Select the top-level node in the Item Master tree.

   The name of this node depends on how you are viewing the Item Master. For example, the node name may be Item Master by Type. For more information, see Section 9.8 on page 9-11.
2. Select **Create > New Item Type,** or click the right mouse button and select **New Item Type** from the shortcut menu.
3. Enter a **Name** and **Description** of the new Item Type.

   To add Properties to the new Item Type, see Section 4.5.5 on page 4-16.

4.5.2 Creating a New Item

To add a new Item to an Item Type:

1. Select an Item Type.

2. Select **Create > New Item**, or use the right mouse button and select **New Item** from the menu.
3. Enter a **Name** and **Description** of the new Item Type.

   To add Properties to the new Item, see Section 4.5.5 on page 4-16.

4.5.3 Changing the Item Type of an Item

You can change an Item Type only if you created the Item Type in Configurator Developer; imported Item Types are read-only.

To change an Item Type:

1. Select an Item in the Item Master.
2. Expand the **Type/Properties** region, then click the **Change** button.
3. Select a new Item Type from the list, or click **New Item Type** to create a new Item Type.
4. Repeat steps 1 through 3 for each Item Type you want to change.

4.5.4 Editing Properties Assigned to Items or Item Types

You can modify a Property assigned to an Item if the Property was created in Configurator Developer. If the Property was imported from Oracle Bills of Material, you cannot modify it.

When you create nodes in the Model structure either by dragging and dropping Items or Item Types from the Item Master or by running a Populator, Configurator Developer maintains a relationship between the Properties on the nodes in the Model structure and the corresponding Items in the Item Master. Therefore, if you modify a Property's value in the Item Master, Configurator Developer updates the corresponding value in the Model structure (although the reverse is not true). However, if you change the Property's value in the Model structure, any future changes that you make to the Property in the Item Master will not be reflected in the Model structure.

To edit an Item Property:

1. Select the Item or Item Type.
2. Click **Edit**.
3. If you selected an Item, modify the **Name**, **Data Type**, or **Default Value**, then click **OK**.

If you selected an Item Type, enter a new Property **Value**, then click **OK**.

4.5.5 Adding Properties to an Item or Item Type

Properties may be imported from Oracle Bills of Material or created in Configurator Developer. You can assign either type of Property to an Item or Item Type, regardless of whether the Item or Item Type was created in Configurator Developer or was imported.

When you assign a Property to an Item, the Property is also assigned to the Item Type. Similarly, when you assign a Property to an Item Type, Configurator Developer assigns the Property to all of its Items. You can then modify the default value for specific Items if the Property was created in Configurator Developer (that is, it is not an imported Property).

To add a Property to an Item:

1. Select the Item or Item Type.
2. Click **Add**.
3. Select the Property to add, then enter a **Default Value**.
4. Click **OK**.

4.5.6 Deleting an Item or Item Type

You can delete an Item or Item Type only if it was created in Configurator Developer. In other words, you cannot delete imported Items or Item Types.

To delete an Item or Item Type:

1. Select the Item or Item Type.
2. Select **Delete** from the **Edit** menu or use the right mouse button to select **Delete** from the shortcut menu.

   You can delete an Item Type only if it contains no items. If you attempt to delete an Item Type that has Items, Configurator Developer prompts you to delete the Items first.
3. Select **OK** to confirm the deletion.

5

Constructing Configuration Rules

One of the most critical activities in constructing your configuration model is to design and construct the rules that govern what the end user can select to make a valid configuration. You need to identify the rules that express relations among the Components, Features, Options, BOM Option Classes, and BOM Standard Items of your Model.

When you defined the requirements for your configurator, you defined the rules for a valid configuration. You now need to determine how you can most effectively and efficiently apply these rules, using the kinds of configuration rules provided by Oracle Configurator Developer. For each type of configuration rule you identify Model elements that are:

- Used as general defaults
- Used as defaults when another option is selected
- Automatically selected when an end user selects another option
- Permitted when an end user selects another option
- Excluded when an end user selects another option

This chapter provides an overview of the rules you can create in Configurator Developer and information about logic states, rule folders and Rule Sequences.

For suggestions about defining rules for best runtime performance, see the *Oracle Configurator Performance Guide.*

5.1 The Configuration Rules Module

Oracle Configurator Developer provides an intuitive, graphical user interface that allows you to drag nodes from your Model and drop them into the appropriate parts of the rule structure. You can create any type of simple configuration rule by

dragging-and-dropping nodes from your Model. When you need to build more complex configuration rules, use the Advanced Expression editor. See Section 5.13, "Advanced Expressions" on page 5-53 for more information.

You can also extend the abilities of Configurator Developer and define custom behavior by creating **Functional Companions**. Functional Companions enable you to create implement custom runtime rules, access information outside the configuration model, perform engineering calculations and integrate with an alternative user interface. See Section 5.14, "Functional Companions" on page 5-67.

5.1.1 Compiling Rules

After you define a rule in Configurator Developer, you must choose **Tools > Generate Active Model** from the Model window to complete the rule definition and ensure each rule is valid within the Model. The Generate Active Model function retrieves data from the database and checks for incomplete rules and inconsistent logic.

You must also generate the Active Model whenever you modify a rule in Configurator Developer to update its definition within the configuration model. At runtime, the Configurator accesses your rules to determine the state of each option before and after the end user makes a selection. The rules alert the end user when an invalid selection is made and selects or deselects other options to ensure a valid configuration is created.

5.1.2 Configuration Rules and Logic State

To enforce the rules you build, the runtime Oracle Configurator maintains a **logic state** for options that the end user can select in the runtime UI. An option in this context is any Model element the end user can select when configuring a product. Options can include an Oracle Configurator Developer Component, Feature, or Option, or any optional BOM Model, BOM Option Class, or Standard Item in the configuration model.

An option's logic state can be True, False, or Unknown:

- A logic state of True means that the option is included in the configuration.
- A logic state of False means that the option is excluded from the configuration.
- A logic state of Unknown means that no decision has been made about the option (for example, it is not selected or, if the option allows a value, it is set to 0).

An option can be added to a configuration or excluded from a configuration either by explicit end user action (for example, selecting an option) or as a consequence of a configuration rule. Oracle Configurator also considers *how* an option is included or excluded from the configuration using variations of the True and False logic states. When an *end user* selects an option, the option's logic state is set to **User True**. When the propagation of a *configuration rule* causes an option to be selected, Oracle Configurator sets the option's logic state to **Logic True**. For example, a Logic rule states "Option A Implies Option B." When the end user selects Option A, Oracle Configurator selects Option B. Therefore, Option B's logic state is set to Logic True.

Similarly, an option may be *excluded* from the configuration in one of two ways. When the end user deselects a previously selected option, the option's logic state is set to **User False**. When an option is excluded by the action of one or more configuration rules, its logic state is set to **Logic False**.

5.1.2.1 Logic State in the Runtime User Interface

An option's logic state determines its appearance in a runtime Oracle Configurator. By default, Configurator Developer indicates logic state using icons. Following is a list of the icons that Configurator Developer uses for each logic state:

- Logic True options appear in the runtime UI with a green check mark
- User True options display with a black check mark
- Logic False options display with a red X
- User False options appear blank
- Unknown options appear blank

By default, options with a logic state of User False or Unknown appear as a blank check box at runtime. An option with either logic state is available to the end user, which means the Oracle Configurator user can select it without receiving a contradiction message.

Displaying available options using the same icon (an empty check box) is appropriate for most end user applications, but you may want to use a specific icon for each so your users can differentiate between options that are User False and Unknown. You can also use different icons for the other logic states. For information about modifying the default logic state icons, see Section 6.3.1.1, "Modifying Default UI Settings" on page 6-25.

When the end user opens a Configurator window, all logic states are initially Unknown, unless you have defined default values (see Section 9.14.3.1, "Initial Value" on page 9-26). The exception is a Feature that has a minimum **Number of**

Selections of one or greater: this has an initial logic state of Logic True because at least one of its Options must be selected to create a valid configuration. The runtime Oracle Configurator keeps track of all changes to an option's logic state that result from an end user's selections and the rules triggered by those selections.

In the DHTML window, a **tooltip** displays when you place the cursor over a logic state icon; tooltip text is not available in the Java applet. Table 5–1 lists the predefined tooltip text that appears in the runtime DHTML UI for each logic state.

Table 5–1 Tooltip Text for Logic State Icons

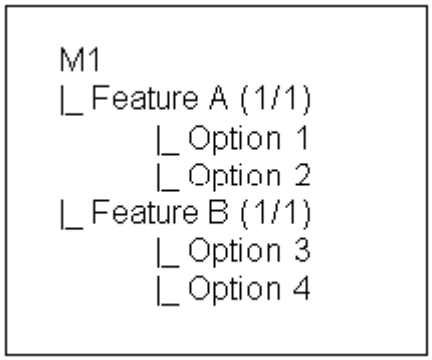
Logic State	DHTML Tooltip Text
Unknown	Not selected
Logic True	Auto selected
User True	User selected
Logic False	Excluded
User False	User deselected

5.1.3 Effectivity and Logic State

Because you can enter an effective date or assign Usages to Model structure nodes in Configurator Developer, there may be times when one or more participants in a configuration rule are not available at runtime. (See Section 3.3, "Effectivity" on page 3-4.) When a node is not available at runtime because of effectivity, it does not appear in the runtime UI, it is considered by the configuration session to have a logic state of Logic False, and any rules in which the node is a participant behave accordingly.

Consider the Model shown in Figure 5–1.

Figure 5–1 Effectivity and Logic State



You assign an effective date to Feature A and create the following Logic rule:

```
Option 1 Requires Option 3
```

At runtime, the date passed to the configuration session does not match the effective date range assigned to Feature A in Configurator Developer. Therefore, Feature A and its Options do not exist in the configuration, and Option 1 and Option 2 are set to Logic False when the configuration session begins. Option 1 is part of the Logic rule mentioned above, so Option 3 is also set to Logic False. See Figure 5–2, "The Requires Relation" on page 5-9.

5.1.4 Unsatisfied Rules

Configuration rules define constraints and requirements for creating a valid configuration and notify the end user when they are violated. When a configuration rule is violated, the configuration is considered invalid, and the end user must make changes before continuing. Configurator Developer also displays a warning message when fewer than the minimum number of options are selected from one or more Features or BOM Option Classes. When this occurs, the configuration is incomplete, and the end user must select additional options to create a complete product.

A third situation can arise when the end user exits a configuration session and one or more options have an Unknown (available) logic state, but additional selections are required to create a complete and valid configuration. When this happens, the rule containing the available options is said to be "unsatisfied."

For example:

A Model contains Options X, Y, and Z. A Logic rule states that "X Implies AnyTrue (Y, Z). " The end user selects X, but neither Y nor Z is selected. At the end of the configuration session, both options are still available and have a logic state of Unknown. As a result, the rule is unsatisfied.

Only Logic rules and Comparison rules can ever be unsatisfied because they are the only rule types that, at the end of a configuration session, may contain options that are required and have an Unknown logic state.

A Logic or Comparison rule can become unsatisfied if it contains:

- A simple expression and specifies All True or Any True of more than one rule participant
- An advanced expression that contains either the AND or OR operator
- The All True or Any True operators with more than one rule participant

5.1.4.1 Examples of Unsatisfied Rules

Example 5–1 Unsatisfied Implies Rule

You define the following Logic rule and provide an unsatisfied rule message:

```
Option A Implies AnyTrue (Not(Option B), Not(Option C))
```

At runtime, the end user selects Option A. Option B and Option C still have a status of Unknown, so the rule is unsatisfied. When the end user clicks **Done**, your unsatisfied rule message appears. The end user returns to the configuration session and satisfies the rule by selecting either Option B or Option C.

Example 5–2 Unsatisfied Requires Rule

You define the following advanced Logic rule and provide an unsatisfied rule message:

```
AnyTrue (Option B, Option C) Requires Option A
```

At runtime, the end user selects Option A. Option B and Option C still have a status of Unknown, so the rule is unsatisfied. When the end user clicks **Done**, your unsatisfied rule message appears. The end user returns to the configuration session and satisfies the rule by selecting either Option B or Option C.

Example 5–3 Unsatisfied Comparison Rule

You define the following simple Comparison rule and provide an unsatisfied rule message:

```
Option A Requires AnyTrue ( ( B < 10), (C > 5) )
```

At runtime, the end user selects Option A. Option B and Option C still have a status of Unknown, so the rule is unsatisfied. When the end user clicks **Done**, your unsatisfied rule message appears. The end user returns to the configuration session and satisfies the rule by entering a value of 1 for Option B.

Example 5–4 Unsatisfied Negates Rule

You define the following advanced Logic rule and provide an unsatisfied rule message:

```
AnyTrue ( Option A, Option B ) Negates Option C
```

You create a second Logic rule with the following definition:

```
 Option Z Negates Option C
```

At runtime, the end user selects Option Z. This gives Option C a status of logic False, but both Option A and Option B are still Unknown. When the end user clicks **Done**, your unsatisfied rule message appears. The end user returns to the configuration session and satisfies the rule by selecting either Option A or Option B.

5.1.4.2 Unsatisfied Rule Messages

By default, Logic and Comparison rules that meet the criteria listed in the previous section do not display a message when they become unsatisfied at runtime. However, when defining the rule you can choose to display a message that contains the rule name, the rule description, or any custom text that you enter. This message is the only way to inform the end user that a rule is unsatisfied and additional options must be selected. If you choose to create a custom message, be sure to provide as much information as possible so the end user knows about the available options and can satisfy the rule by making additional selections.

Note that there may be times when a rule is unsatisfied but it is not necessary to display a message to the end user. For example, you define the following Logic rules:

1. All True (A, B) Implies C
2. X Excludes C

At runtime, the end user selects X, which makes C false. To strictly satisfy Rule 1, at least one of (A, B) would have to be false. In other words, if both A and B are Unknown, Rule 1 is unsatisfied. However, the end user probably does not care

whether A, for example, is false or Unknown. Therefore, Rule 1 should have the Unsatisfied Message attribute set to None.

In general, you should not specify an unsatisfied message when the rule might be unsatisfied because an option is *Unknown* rather than false.

Creating an Unsatisfied Rule Message

After defining the Logic or Comparison rule, perform the following to create an unsatisfied rule message:

1. Expand the **Unsatisfied Message** attribute.
2. Specify the information to display at runtime when the rule is unsatisfied by selecting the **None**, **Rule Name**, **Rule Description**, or **Custom** radio button.

   Select Custom to enter detailed information about the rule, such as the available options that may cause it to be unsatisfied. See Section 9.18.4, "Unsatisfied Message" on page 9-38.

5.1.5 Logical Relationships

You can express constraints among elements of your Model in terms of logical relationships. For example, one Feature may require or imply that another Feature or Option be included in the configuration. Oracle Configurator Developer provides the following logical relationships:

- "Requires" on page 5-9
- "Implies" on page 5-9
- "Excludes" on page 5-10
- "Negates" on page 5-10
- "Defaults" on page 5-10

The following five sections describe these relationships and present tables illustrating their action.

In each row of the table, the unshaded portion indicates the logic state chosen by the end user, and a shaded portion indicates the logical state that results from the selection. The arrows indicate the direction in which the logic state is propagated. Notice that logic state can propagate both from the A side to the B side of the relation, and from the B side to the A side. Notice also that for some values and some logic relations, there is no propagation of logic state.

The text following the table describes the effects of applying that type of relationship.

5.1.5.1 Requires

The effect of the Requires relation: If you set either option to true or false, the corresponding option in this relation is also set to the same state (true or false). The two are forced to be the same.

Figure 5–2 The Requires Relation

A	REQUIRES	B
True	→	True
False	→	False
True	←	True
False	←	False

5.1.5.2 Implies

The effect of the Implies relation: If you set A to true, then B is set to true as well; conversely, if you set B to false, then A is also set to false.

Compare this diagram to the corresponding one for the Requires relation. In the Implies relation, note that there are no “pushing” arrows when a result is undetermined.

Figure 5–3 The Implies Relation

A	IMPLIES	B
True	→	True
False		Undetermined
Undetermined		True
False	←	False

5.1.5.3 Excludes

The effect of the Excludes relation: If you set an option to true, the corresponding option in this relation becomes false, since it is not allowed. If you set an option to false, it has no effect on the other option.

Figure 5–4 The Excludes Relation

A	EXCLUDES	B
True	→	False
False		Undetermined
False	←	True
Undetermined		False

5.1.5.4 Negates

The effect of the Negates relation: If you set an option to true or false, the corresponding option in this relation is immediately set to the opposite. The two are forced to be opposed.

Figure 5–5 The Negates Relation

A	NEGATES	B
True	→	False
False	→	True
False	←	True
True	←	False

5.1.5.5 Defaults

The effect of the Defaults relation: If you set the A side of the relation to true, the B side is set to true. Unlike other logic relations, the logic state of the B side is not enforced. You can set the B side to true or false regardless of the state of the A side.

The relations only action is to set B to true when A is set to true, if B is not true already. You can think of Defaults as defining additional suggested selections if the end user makes a certain selection.

The Defaults relation can be used to set up an initial configuration, for example, by triggering a set of Defaults relations with a Boolean Feature whose initial value is true. Options that are Logic False because of a Defaults rule appear as though they have a logic state of Unknown in the runtime UI to indicate that they can be selected by the end user.

For example, Feature X has three Options: Option A, B, and C. The maximum **Number of Selections** for Feature A is 2, and you define the following Logic rule:

```
Feature X Defaults AllTrue (A, B)
```

At runtime, both Option A and B are selected and have a logic state of Logic True (indicated by a green check mark). Even though Option C has a logic state of Logic False (because the maximum Number of Selections for Feature X has been reached), it appears blank in the runtime UI. (This example assumes you are using the default logic state icons.)

> **Note:** Using many Defaults Logic rules can significantly affect performance of your configuration model at runtime. For more information, see the *Oracle Configurator Performance Guide*.

5.1.5.5.1 Defaults Rules and Logic States There may be times during a configuration session when an option is deselected as the result of a Defaults Logic rule, and the option's logic state changes to User False. If you are using the same icon to represent both Unknown and User False logic states, Oracle Configurator end users will not be able to distinguish between the two states at runtime. As a result, your end users may find it more difficult to create a valid configuration.

To avoid this situation, you may want to display Unknown and User False logic states differently by specifying a different icon for each. For more information, see Section 6.3.1.1, "Modifying Default UI Settings" on page 6-22.

5.1.6 All True and Any True

The preceding relations describe logical relationships in terms of true and false values for A and B sides of the relation. You can also build logic relations where the A or B side involves more than one term, for example, all Options of a Feature. In

this case, you must specify whether that side of the relation is true if ALL of the terms are true, or if that side is true if ANY of the terms are true.

- Select **All True** in the **Definition** section for the rule, or use the AllTrue function in the Advanced Expression editor, if you want the side of the relation to evaluate to true only if all terms are true. If any term is false, the expression is false. This is a logical AND expression.
- Select **Any True** in the **Definition** section for the rule, or use the AnyTrue function in the Advanced Expression editor, if you want the side of the relation to evaluate to true if any term is true. The expression is false only if all terms are false. This is a logical OR expression.

For information about NotTrue, see Section 5.13.3, "Using NotTrue in Advanced Expressions" on page 5-66.

5.1.7 Summary of Logical Relationships

Compare and contrast the effects of the logical relationships as shown in Table 5–6:

Figure 5–6 Summary of Logical Relationships

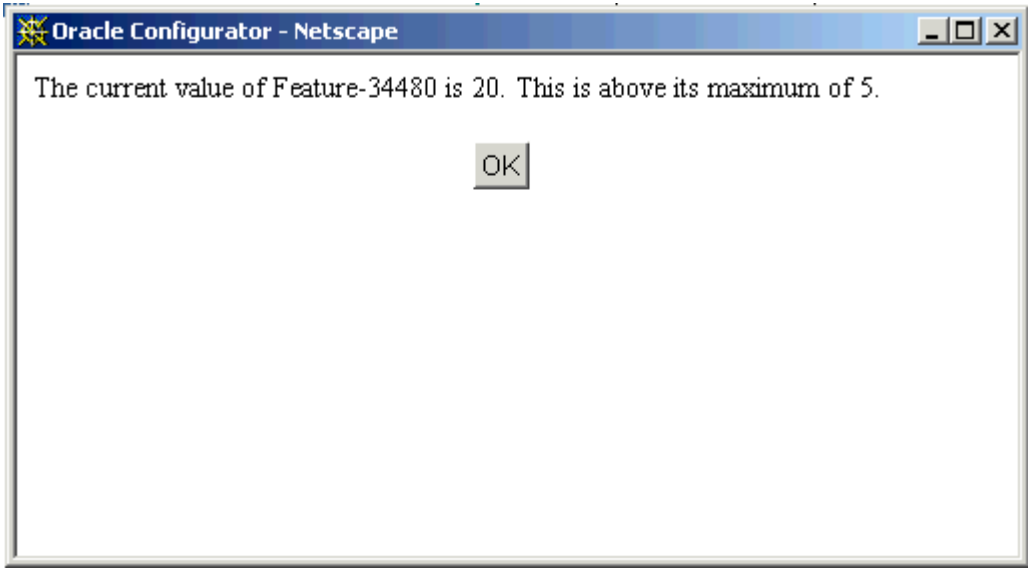
A	REQUIRES	B	A	IMPLIES	B
True	→	True	True	→	True
False	→	False	False		Undetermined
True	←	True	Undetermined		True
False	←	False	False	←	False
A	**NEGATES**	**B**	**A**	**EXCLUDES**	**B**
True	→	False	True	→	False
False	→	True	False		Undetermined
False	←	True	False	←	True
True	←	False	Undetermined		False

5.2 Enforcing Logical Relationships

The rules you create in Oracle Configurator Developer shape choices the end user can make in the deployed configuration model. If the end user makes a selection that violates a rule, the runtime Oracle Configurator displays a message that describes the effect of the selection on the configuration.

For example, if the end user sets a numeric Feature to a value that exceeds its defined maximum, the system responds with a message similar to the one shown in Figure 5–7.

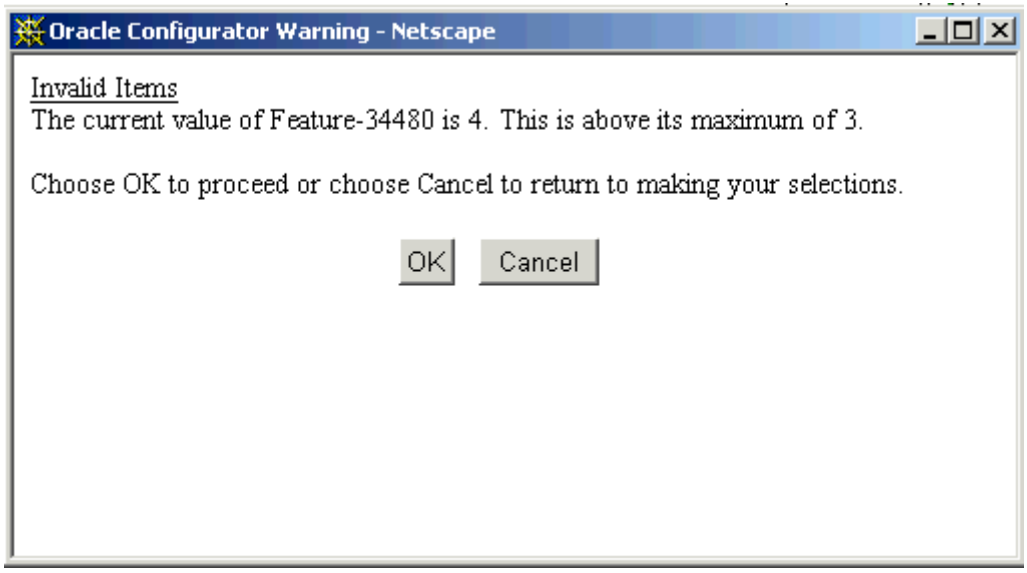
Figure 5–7 Maximum Exceeded Message



When this occurs, the end user must enter a new value for the Feature before saving the configuration.

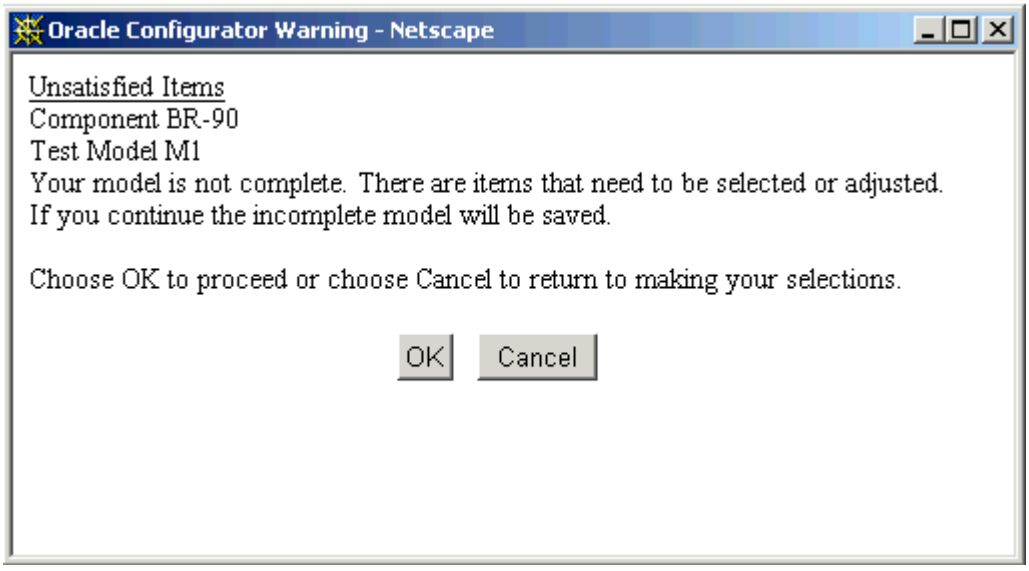
However, Numeric rules are handled differently. If the user selects values that cause a numeric Feature or Option count to exceed its defined maximum or minimum values, the value of a Total is exceeded, or a Resource is overconsumed, the system responds with a message similar to the one shown in Figure 5–8.

Figure 5–8 Invalid Configuration Message



If the end user attempts to save the configuration in this state, or any other state that violates the rules you have established, the configuration model responds with a message similar to the one shown in Figure 5–9.

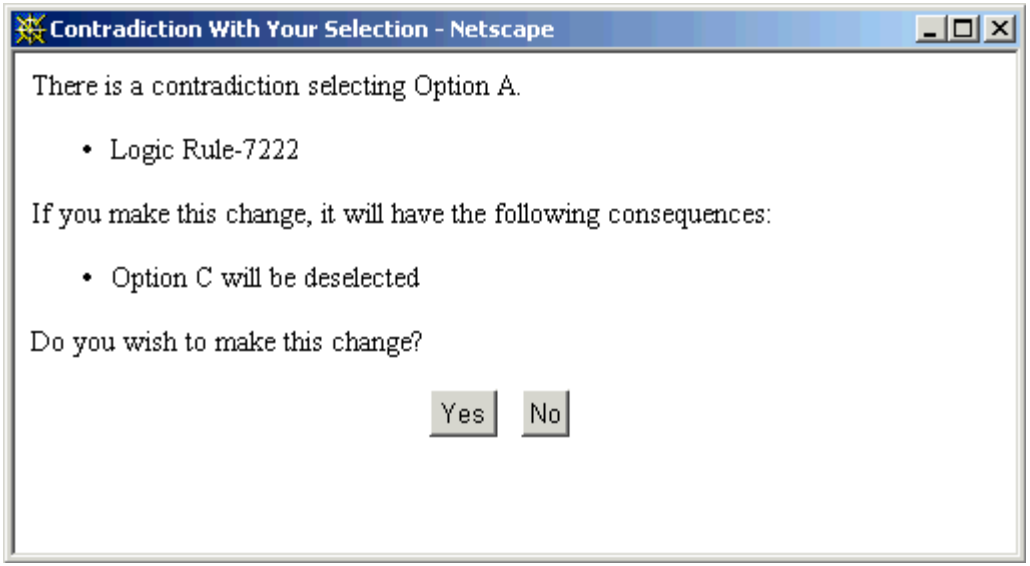
Figure 5–9 Unsatisfied Configuration Message



The end user can save the invalid configuration by clicking **OK**.

If the end user attempts to make a change to the configuration that violates one or more other rules, the systems responds with a contradiction message similar to the one shown in Figure 5–10.

Figure 5–10 Contradiction Message



The end user must either agree to reject the last selection or indicate that the change should be made. In the latter case, the system changes values of other Features, Options, and so on until the configuration is restored to a valid state.

If a Feature or Option has a min / max of 1 and an end user's selection triggers a rule that changes its state to Logic False, the item's icon appears with a red X in the runtime UI. When an end user selects an item marked with a red X, Configurator Developer displays a contradiction message and the end user must click **OK** or **Cancel** before proceeding. Features or Options with a min / max of 1 that are User False (that is, set to false by an end user's selection, rather than by a configuration rule) do not appear with a red X and the end user can select them without receiving a contradiction message.

You can prevent a red X from appearing at runtime by setting the **Hide when unselectable** option to Yes on the UI root node in the UI module. By doing this, Features and Options no longer appear in the runtime UI when their logic state changes to Logic False. See Section 6.1.5.1, "Dynamic Visibility" on page 6-11.

For more information about logic state icons, see Section 6.3.1.1, "Modifying Default UI Settings" on page 6-25.

5.3 Overriding User Selections in the Runtime UI

At runtime, there may be times when the system deselects one of an end user's previous selections but does not alert the end user by displaying a notification. The system automatically deselects an option in the runtime UI when both of the following are true:

- An end user selects an option that causes the option to exceed its maximum quantity
- Any one of the previous selections was made by the end user (that is, its logic state is User True)

However, the system cannot deselect a selection that was made by the action of a rule (that is, its logic state is Logic True). If none of the previous selections were made by the user, no override occurs and a the system displays a violation message in the UI.

For example, an Option Feature called f1 has a minimum of 1 and a maximum of 2. This Feature contains 3 Options: o1, o2, and o3. The maximum of f1 is reached (2 options are selected) either by end user selection or by the action of a rule. The third option is Unknown and available. When the end user selects the third option, whether the system overrides a previous selection depends on how the selections were made.

Consider these examples:

- If o1 and o2 are User True and o3 is made true, the system deselects the most recent end user selection and selects o3.
- If o1 is User True and o2 is Logic True and the user selects o3, the system deselects o1.
- If o2 is being contributed to from another item, both o1 and o2 are User True, and the user selects o3, the system deselects o1.
- If both o1 and o2 are Logic True and the user selects o3, there is a contradiction and o3 cannot be made true. In this case, the system displays a violation message.

For more information on logic states, see Section 5.1.2, "Configuration Rules and Logic State" on page 5-2.

5.4 Types of Configuration Rules

Table 5–2 summarizes each type of configuration rule.

Table 5–2 Configuration Rule Types

Rule Type	Description
Logic Rules on page 5-21	Define logical relationships between Features, Options, BOM Option Classes, and BOM Standard Items using a *logic relations*. The logic relations available are: Requires, Implies, Excludes, Negates, and Defaults.
Numeric Rules on page 5-21	Perform a numeric operation on one or more numeric Features, Option counts, or Totals placing the result in a numeric Feature, Option count, Total, or Resource. The rules available are: Contributes to and Consumes from, each with a variety of ways in which to specify the numeric operation.
Comparison Rules on page 5-29	Perform a comparison between the value of a Property of one or more Features and the value of another such Property, or some constant value.
Property-based Compatibilities on page 5-32	Specify matches between the Options of one or more Features that have a common Property.
Explicit Compatibilities on page 5-35	Specify matches between the Options of one or more Features in explicit tabular form.
Design Charts on page 5-40	Specify compatibility matches between one Primary Feature's Options and multiple Secondary Feature's Options in explicit tabular form.
Rule Sequences on page 5-47	Specify an ordered set of rules whose effectivity dates are set so that a rule in the sequence becomes effective at the same time its predecessor ceases to be effective.
Functional Companions on page 5-67	Write code to perform pre-selection or validation functions that go beyond Oracle Configurator Developer's supplied functionality.

To create configuration rules for an element of your Model, perform the following:

- Determine the type of rule you want to create
- Build the rule expression
- Generate the rule logic
- Test the rule (see Chapter 7 on page 7-1)

5.4.1 Rule Folders

Configurator Developer provides a folder called *Model Name* Rules to organize your configuration rules. You can view rules by type, by folder, or by name, but this folder appears in the Configuration Rules view regardless of the method selected. View rules by type to display your rules in sub-folders according to the type of rules they contain (for example, Logic rules, Numeric rules, and so on). When viewing by name, all rules appear alphabetically by name, not in folders. By default, Configurator Developer displays configuration rules by folder.

You can also organize rules by creating your own folders and providing descriptive names to group them. When you create a rule, Oracle Configurator Developer places a node for the rule in the folder you selected. You can create as many rule folders as you need to organize the rules in a meaningful manner.

Rule folders that you create can contain any type of rule. You can use standard drag-and-drop or cut, copy, and paste operations to move or copy a rule from one folder to another. However, the same rule cannot reside in more than one folder. Copying a rule to a different folder creates a new, separate rule that can be modified independently of the original.

To create a rule folder:

1. Go to the Configuration Rules module.
2. Select an existing folder (for example, *Model Name* Rules).
3. Choose **Create > New Folder** or right click and select **New Folder**.

5.4.2 Enabling and Disabling Rules

All rules except Functional Companions can be enabled or disabled. The Attributes View for each rule contains a check box labeled **disable** to the right of the rule name. Select this box to disable the rule. A check mark appears to indicate that the rule is disabled. Select the box again to enable the rule.

You can also disable and enable all the rules in a selected folder by selecting the appropriate button in the Attributes View for the folder. Disable and enable at the folder level are merely convenient methods to change the setting for a group of rules. You can still enable and disable individual rules within the folder.

Enabling and disabling rules can be a useful tool in testing and debugging your configuration model.

5.5 Building Configuration Rules

Follow these general steps to build any of the available configuration rules. For detailed information on building a specific type of rule, see:

- "Logic Rules" on page 5-21
- "Numeric Rules" on page 5-21
- "Comparison Rules" on page 5-29
- "Property-based Compatibilities" on page 5-32
- "Explicit Compatibilities" on page 5-35
- "Design Charts" on page 5-40
- "Rule Sequences" on page 5-47
- "Functional Companions" on page 5-67

1. Select the **Rules** button on the main toolbar. The Configuration Rules tree appears in the lower-left pane.
2. Select a node in the Configuration Rules tree.
3. Open the **Create** menu and select the **New** command for the type of rule you want to create. You can also highlight the Configuration Rules node in the Context Tree View, select the right mouse button, and select **New** command from the pop-up menu.

   Note: The root node of a BOM Model cannot be a participant in a configuration rule. This is because the root node of a Model does not have an associated count or logic state.

4. Enter the following attributes for the rule.
   - "Name" on page 8-13
   - "Description" on page 8-13
   - "Parameters" on page 9-37 (Compatibility rules only)
   - "Definition" on page 9-38
   - "Violation Message" on page 9-38
   - "Unsatisfied Message" on page 9-38 (Logic rules only)

– "Effectivity" on page 9-31

5. Select **Tools > Generate Active Model** when you are ready to test your rules.
6. Test your new rule. See Section 7, "Testing and Debugging" on page 7-1 for details about testing your configuration rules.

5.6 Logic Rules

Logic rules establish a logical relationship between one part of your Model structure and another. You can use them to control the selections the end user makes.

5.6.1 Building Logic Rules

For a general description of creating configuration rules, see Section 5.5 on page 5-20.

1. Expand the **Definition** attribute.
2. Drag and drop the specific Feature, Option, BOM Option Class, or BOM Standard Item nodes from the Model View to the rule element A and B lists.
   - To apply the rule to a Feature, Option, BOM Option Class, or BOM Standard Item, select the node with the **left** mouse button and hold down the button while you drag the node to the rule element list.
   - To apply the rule to all of the Options of a Feature or BOM Option Class, select the node with the **right** mouse button, drag the node to the rule participant field, then release the mouse button. Then, choose **OptionsOf(nodename)** from the menu.
   - Indicate whether any of or all of the elements in the element lists must be true for the rule by selecting **Any True or All True**. See Section 5.4, "Types of Configuration Rules" on page 5-17.
3. Indicate whether the result of rule element A Requires, Implies, Excludes, Negates, or Defaults rule element B.

   For a description of each rule type, see Section 5.1.5, "Logical Relationships" on page 5-8.

5.7 Numeric Rules

Numeric rules express constraints between elements of your Model in terms of numeric relationships. With Numeric rules, end user selections can contribute to or

consume from a Resource, Total, numeric Feature, Option count, or the minimum or maximum number of component Instances allowed in a runtime Oracle Configurator.

5.7.1 Contributes to

A **Contributes to** Numeric rule specifies addition of a value to a specified numeric Feature, Option count, Total, Resource, and the minimum or maximum number of Instances. Calculation of the numeric value can involve Constants, Boolean values, numeric Features, Option counts, Option Properties, Totals, the Minimum and Maximum number of Instances and the Instance count.

5.7.2 Consumes from

A **Consumes from** rule specifies subtraction of a numeric value from a specified numeric Feature, Option count, Total, Resource, or Instance count. Calculation of the numeric value can involve Constants, Boolean values, numeric Features, Option properties, Totals, the Minimum and Maximum number of Instances, and the Instance count.

5.7.3 Using Numeric Features in Numeric Rules

Although it is not common practice, it is possible to define a Numeric rule in which a Numeric Feature contributes to or consumes from another Numeric Feature. (In other words, the participants on *both* sides of the rule are numeric Features.) If you define rules such as this, it is important to understand that the runtime behavior may produce unpredictable results. This is because:

- Oracle Configurator calculates the values for the participant Features in a way that is not apparent to the end user (this is explained below).
- The end user can enter a value for either Feature at any time.

Consider the following rule in which both participants are Numeric Features:

```
Feature A * (Constant 1) Contributes to Feature B
```

At runtime, an end user enter 10 for Feature A, which sets Feature B to 10. Later in the configuration, the end user changes Feature B by entering a value of 20 (this has no effect on Feature A). Then the end user changes the value of Feature A to 30. When this happens, Oracle Configurator sets Feature B to 40.

In the example above, Oracle Configurator initially sets Feature B to 0 (zero). However, when an end user manually enters a value for the Feature on the **B** side of

the rule, the system also maintains an *internal* value for the Feature. This internal value is not visible to the end user and is determined by subtracting the Feature's old value from the new value (that is, new value - old value). Therefore, when the end user enters 20 for Feature B, the *internal* value becomes 10. Then, when the end user changes Feature A to 30, Oracle Configurator contributes 30 to Feature B's internal value (10) which produces a result of 40.

This situation may confuse the end user and might not be the behavior you expected when defining the rule. Therefore, it is generally better practice to use a Total or a Resource on the **B** side of Numeric rules, since these node types are read-only at runtime and will not produce the behavior described above (because the end user cannot enter a value for the Feature on the **B** side of the rule).

The behavior described above occurs whether the rule uses the **Consumes from** or the **Contributes to** relation.

5.7.4 Building Numeric Rules

This section describes how to build simple Numeric rules. For information about defining more complex Numeric rules, see Section 5.13, "Advanced Expressions" on page 5-53.

For a general description of creating configuration rules, see Section 5.5 on page 5-20.

To build a Numeric rule:

1. Expand the **Definition** attribute.
2. The **A** side of the rule consists of a comparison operator and two operands. Select one of the following operators from the list:
   - * Multiplication
   - / Division

> **Note:** Numeric rules containing the division operator *[/]* propagate a 0.0 result when the divisor is 0. Numeric rules containing the division operator /result in an error when the divisor is 0. For example, x [/] y produces a 0.0 result when the divisor is 0, whereas x/y produces an error when the divisor is 0. New rules cannot be created using the [/] division. See Section 5.7.4.1, "Examples of Numeric Rules Using the Division Operator" on page 5-25.

The operands can be one of:

- **Boolean Expression:** A Boolean expression is one that can have a True or False value. A value of True is interpreted as a one (1) and a value of False is interpreted as a zero (0). Therefore, use **Boolean Expression** as the first operand to contribute either a one or a zero, depending on the logic state of the Feature or Options in the rule. For example, you might define a rule that states "when Feature1 is selected, contribute a value of 5 (Feature1 x a constant value of 5) to the Weight Total."

  Indicate whether **any** or **all of** the Features or Options in the list must be true for the rule by selecting **Any True** or **All True**. See Section 5.1.6, "All True and Any True" on page 5-11.

- **Option Count:** Option Count(s) use the number of Options selected at runtime in the rule. For example, if an end user selects 3 Options, a value of 3 is used in the rule. An option that is *not* selected or has a value of 0 (zero) does not contribute a value to the rule. (In this case, the option's logic state is Unknown. See Section 5.8 on page 5-28 for more information.)

  If the other operand is an Option Property, the respective Property value of each Option is contributed. For example, the count of each selected Option is multiplied by the respective value of the Option's "Weight" Property and is contributed to a Total called "Total Weight." For more information about Counted Options, see Section 9.14.3, "Type" on page 9-23.

  Note: If you choose Boolean Expression or Option Count, you can drag and drop nodes to participate in the rule using either the **left** or **right** mouse button. However, these buttons perform different actions depending on the type of node that you select. For example, to include only a specific Feature or BOM Option Class as a participant in the rule, select the node using the **left** mouse button, then drag and drop the node in the rule element list. To include *all* of a Feature's Options as participants in the rule (or all Standard Items within a BOM Option Class), select the node using the **right** mouse button, drag the node to the rule element list, and release the button. Then choose **OptionsOf** from the menu.

- **Constant:** Enter either a positive or a negative numeric constant.
- **Total or Numeric Feature:** Select the specific Total or Numeric Feature with the **left** mouse button, hold the button and drag the node to the rule

participant field. For more information about Totals and Features, see Section 4.2, "Model Structure" on page 4-2.

3. Indicate whether the result of the **A** side of the rule **Contributes to** or **Consumes from** the participant on the **B** side of the rule.

4. Drag and drop a node or node Property to the **B** side of the rule. To use a Total, Resource, or Numeric Feature node, select and drag it using the **left** mouse button. To use a node Property (such as the Option Count, BOM Standard Item count, or Minimum or Maximum Instances of a Model or Component), select and drag the node using the **right** mouse button. (See Section 5.7.5, "Building Rules Using Node Properties" on page 5-26.)

   Note that any combination of Numeric rules that results in a negative contribution to an Option count, BOM node count, or count Feature (a Numeric Feature with a non-negative minimum), has no effect at runtime. This situation can arise in various ways. For example:

   - A single Numeric rule consuming a positive value
   - A single Numeric rule contributing a negative value
   - Multiple Numeric rules that together result in a negative net contribution

   In some cases, you may have other rules that offset a negative contribution. In these instances, the rule is valid and contributes in the normal way. See Section 5.7.6, "Negative Contributions" on page 5-27.

5.7.4.1 Examples of Numeric Rules Using the Division Operator

Example 5–5 Numeric Rule Using the Division Operator

Consider two Features Fa and Fb, a Total T and the following division rule:

Fa.count divided by Fb.count contributes to T

If the division rule maps to the division operator ([/]), then setting the count of Fb.count to 0 (or setting Fb to false) will propagate 0.0 for T.

If the division rule maps to the division operator (/), setting the count of Fb.count to 0 (or setting Fb to false) will report an Invalid Domain exception.

Example 5–6 Another Numeric Rule Using the Division Operator

Consider a Total T1 with an initial value of 0, a Total T2 with no initial value, a constant C with a value of 2 and the following rule:

```
C divided by T1 contributes to T2
```

If the division rule maps to the division operator [/], then activation of the rule propagates 0.0 for T.

If the division rule maps to the division operator /, activation of the rule reports an Invalid Domain exception.

5.7.5 Building Rules Using Node Properties

All Model nodes have both User and System Properties and you can use these Properties when building certain types of configuration rules. You may want to use these Properties to implement additional kinds of constraints in your Model. For more information about User and System Properties, see Section 3.2 on page 3-2.

You typically use the Advanced Expression editor to build a rule using a node's User Properties, System Properties, or both. The Advanced Expression editor is available when defining any type of Logic rule, a Property-based Compatibility rule, the **A** side of a Numeric rule, or the **B** side of a Comparison rule. To see how User and System Properties appear in the Advanced Expression editor, see Figure 5–18 on page 5-57.

You can use User Properties when building a Numeric rule by right-clicking on the node in the Model structure and dragging it to the **B** side of the rule. When you do this, note that only Properties that can logically be contributed to or consumed from are available. For example, if you are dragging and dropping an Option, only the System Property "Count" is available, since it does not make sense to contribute to or consume from the "Name" Property.

Example 5–7 Contributing to the Count of an Option

Each time an Option from Feature X is added to the configuration, you need to ensure that the quantity of Option Y increases by 1. To create this rule, drag and drop Feature X to the **A** side of the rule, then right-click on Option Y in the Model structure and drag it to the **B** side of the rule to add the "Count" System Property. The Numeric rule you build to do this has the following definition:

```
OptionsOf{Feature X} * Constant 1, Contributes to Option Y.#Count
```

Note that in this example, the **Counted Options** setting must be selected for the Feature to which Option Y belongs; otherwise, Option Y is simply set to True when an Option from Feature X is selected since it has no numeric value.

Example 5–8 Contributing to the Maximum Number of Component Instances

When the end user orders a specific quantity of Integer Feature A, you want to modify the number of instances of Model Z that are allowed in the configuration. When the configuration session begins, Model Z has **Maximum Number of Instances** set to 3. To change the number of instances allowed at runtime, define a Numeric rule in which Feature A contributes to the MaximumInstances System Property. To use this property in the rule, right-click and drag Model Z to the **B** side of the rule, then select **#MaximumInstances** from the menu.

The Numeric rule in this example has the following definition:

```
Total or Numeric Feature{Feature A} * Constant 1,
Contributes to Model Z.#MaximumInstances
```

If the end user specifies a quantity of 6 for Feature A, the maximum number of instances of Model Z allowed in the configuration dynamically changes to 9 (that is, 6 plus the initial setting of 3).

Note: You cannot create this type of rule for an ATO BOM Model that is a child of another ATO BOM Model.

Note: Rules like the ones described in this section affect only the minimum and maximum number of instances allowed in a runtime Oracle Configurator; they do *not* change a component's **Instances Minimum** or **Maximum** values in Oracle Configurator Developer.

5.7.6 Negative Contributions

Negative contribution include both a Consumes from rule with a positive value, and a Contributes to rule with a negative value. If the sum of contributions is a negative value, it is ignored. If the sum of contributions is positive, the result is contributed in the ordinary way. For example, if you have a counted option with three rules contributing 5, 4, and -3, the value contributed is 6.

Contributions to a BOM Model are handled similarly. For example, you have a BOM Model with a Quantity count of 3, and a child BOM Option Class with a Quantity count of two, and three rules contributing 5, 4, and -3, to the BOM Option Class. The contribution is calculated as:

```
   5    +    4    +   (-3)    =       6
(rule1)     (rule2)    (rule3)     (contributions)
```

The final count of the BOM Option Class is calculated:

```
Floor(      6     /     3     )     *     3     =     6
   (contributions) (parent count)    (parent count)
```

The sum of contributions (6) is exactly divisible by the parent count (3), so the final result is 6.

If the sum of contributions is not exactly divisible by the parent count, the result is the closest smaller exact multiple of the parent count. For example, if the contributions are 5, 4, and -4, the result is 3.

If the sum of contributions is smaller than the parent count, then the result becomes the default count. For example if the rule contributions are 5, 4, and -7, the result is 6 (which comes from 2 * 3).

If the sum of contributions is a negative value, it is ignored.

5.8 Unknown Values and Rule Propagation

When the end user starts a new configuration in the deployed Oracle Configurator, many elements in the Model structure have a logic state of Unknown until the user makes a selection. There are several situations in which Unknown values in rule element A of a Numeric Rule cause the rule not to propagate, so the rule has no effect.

If all of the participants in rule element A are Unknown, the rule does not propagate.

If the expression in rule element A involves the minimum or maximum number of component instances, or the operators '+' or '-', the rule propagates when the end user makes a selection so that one participant in rule element A is no longer Unknown. For example, you create the following rule:

`Min(A, B) = 500`

At runtime, the end user selects option A, but option B is still Unknown (not selected). As a result, A is set to 500. If neither A nor B is selected, the result of the expression is Unknown.

If the expression in rule element A involves the operators '*' or '/', the rule propagates if one rule participant is not Unknown and has a value of 0, or if all participants are not Unknown.

An integer Feature (Data Type is either Integer Number or Decimal Number) can have a value of 0 (zero) and be Unknown if its **Minimum** is greater than or equal to zero.

Numeric Rules involving Property values do not propagate unless the option with the Property is selected.

When any node that can have a logic state is a participant in a rule and the value of the node is 0, its logic state is Unknown and the rule does not propagate until its value changes. For example, you define the following rule:

```
Option1 = 0 Negates Feature2
```

This rule does not propagate when Option1 is set to zero, regardless of whether it is set to zero by default (that is, it has a value of 0 when the configuration session begins) or if it is set to zero by the end user at runtime.

For a list of all nodes that have a logic state at runtime, see Section 5.1.2, "Configuration Rules and Logic State" on page 5-2.

5.9 Comparison Rules

Use Comparison rules to compare two numeric values to produce a logical result. Two numeric values are compared to determine if the first value is greater than, greater than or equal to, less than, less than or equal to, equal to, or not equal to the second value. The result of this comparison is a logical value (true or false). For example, the numeric value of one product Feature or Option compared to the numeric value of another may require or imply that another Feature or Option be included in the product configuration.

When defining Comparison rules, it is recommended that you avoid rules that raise contradictions. Instead, construct your rules so they lead to a resource violation. Rules that raise contradictions are much less flexible because they require one or more previous selections to be deselected before the end user can continue. A resource violation, however, only displays a warning indicating that a specific value has been exceeded, does not deselect any options, and allows the end user to acknowledge the warning and proceed with the configuration. (See Example 5–9 on page 5-30.)

Comparison rules that raise contradictions can also lead to problems caused by an **intermediate value**. Oracle Configurator triggers each rule in a configuration model sequentially (that is, one after another), rather than propagating all rules simultaneously. As a result, a rule may be violated when it reaches a specific value, even if you defined another rule with the intention that it would prevent the

violation from occurring in the first place. However, since the intermediate value is reached before the other rule propagates, a violation occurs. (See Example 5–9 on page 5-30.)

Example 5–9 Best Practices when Constructing Comparison Rules

Model X contains the following:

- Feature D, which is an Integer Feature with no **Minimum** and no **Maximum** values.
- Always True, which is a List of Options Feature with both the **Minimum** and **Maximum Number of Selections** set to 1.

  The "Always True" Feature is commonly used within configuration models to provide a constant within a configuration.

Model X also contains the following rules:

```
Rule 1: Option A Requires AllOf (Option B and Option C)
Rule 2: Option B Contributes 2 to D
Rule 3: Option C Contibutes 2 to D
Rule 4: Feature D < 4 Negates Always True
```

At runtime, the end user selects Option A. Rule 1 propagates and causes both B and C to be selected. Remember that Oracle Configurator does not trigger rules sequentially, so these rules will not propagate in the order listed above. It is therefore likely that Rule 4 will be violated before the value of D reaches 4 (see Rule 4). This is the problem with intermediate values mentioned at the beginning of this section. Additionally, since Rule 4 raises a contradiction, it requires that either B or C be deselected before the end user can continue.

The recommended method is to design your Model and rules so that instead of a contradiction, a resource violation occurs. Using the example above, you can do this by specifying a **Minimum** value of 4 for Feature D, and then deleting Rule 4. Then, when an end user selects Option A, a warning appears (instead of a contradiction message); the user can then acknowledge the message and continue.

5.9.1 Building Comparison Rules

For a general description of creating configuration rules, see Section 5.5 on page 5-20.

1. Expand the **Definition** attribute.

2. The A side of the rule consists of a comparison operator and two operands. Select an operator from a dropdown list. The operator can be one of:

   - <> not equal
   - = equal
   - > greater than
   - **<** less than
   - < less than or equal to
   - > greater than or equal to

   Select the type of operand from a dropdown list located above the operand field. The operands can be one of:

   - **Constant:** Type a positive numeric constant in the operand field.
   - **Total or Numeric Feature:** Select a Total or Numeric Feature with the **left** button and drag it to the operand field. Only one Total or Numeric Feature can be selected for each operand field.
   - **Option Count:** Select an Option or BOM Standard Item with the **left** button and drag it to the operand field. Only one Option Count can be selected for each operand field. See Option Count in Section 5.7.4, "Building Numeric Rules" on page 5-23.
   - **Option Property:** Select an Option or BOM Standard Item with the **left** button and drag it to the operand field. Select a Property from the dropdown list that appears below the **operand** field.

3. Indicate whether the result of rule element A Requires, Implies, Excludes, or Negates rule element B.

4. Drag and drop the specific Feature, Option, BOM Option Class or BOM Standard Item nodes from the Model View to the rule element B.

   Indicate whether to apply **Any True** or **All True** to the Features or Options in this list. See Section 5.4, "Types of Configuration Rules" on page 5-17.

5.10 Compatibility Rules

Compatibility rules include Property-based Compatibilities, Explicit Compatibilities, and Design Charts. Each type of Compatibility rule defines what items are allowed in a configuration, when some other item is selected. Property-based Compatibilities define the compatibility relationships in terms of

the values of Properties shared by Features or BOM Option Classes. Explicit Compatibilities and Design Charts provide two different formats to define specifically what Features or Options Classes are compatible.

It is important to remember that, unlike Logic rules, Compatibility rules do not select anything. They do not drive an option's logic state to true, but they do set logic state to false for Options that are incompatible with an end user's selections. If the Features or Option Classes involved in a compatibility relationship have a **Minimum Number of Selections** of 1, then when all options but one are set to false the remaining option is set to true. Similarly, selection of one option of such a Feature drives the other options to false. The action of Compatibility rules is most clear when the Features involved have a **Maximum Number of Selections** of 1. The behavior of the rules is more complex and may be confusing if the Maximum is greater than 1.

For example, OC1 and OC2 are Option Classes that have a **Minimum** and **Maximum Number of Selections** set to 1 and 3, respectively. OC1 has three Standard Items: S1, S2, and S3. OC2 also has three Standard Items: S4, S5, and S6. You define a Compatibility rule that includes OC1 and OC2 as participants, and specify the following compatibilities:

- S1 and S4
- S2 and S5
- S3 and S6

However, because the **Maximum Number of Selections** for both Option Classes is 3, the Oracle Configurator end user is not limited to only one compatible selection for each option; in fact, the rule allows *any* combination of its participants.

Note: Oracle Configurator Developer optimizes Compatibility rules internally, which allows it to process them faster than any other type of configuration rule.

For important information about the default runtime behavior of Compatibility rules, see Section 5.10.3, "Gated Combinations" on page 5-37.

5.10.1 Property-based Compatibilities

A Property-based Compatibility rule specifies valid combinations of Feature Options or BOM Standard Items based on Property values. Many types of Model

structure nodes can have Properties, but only Features of type List of Options and BOM Option Classes can participate in a Property-based Compatibility rule.

Property-based Compatibility rules that use BOM Option Classes and Features require much less maintenance than Explicit Compatibility rules since you can update each rule automatically simply by reimporting (refreshing) the BOM.

If you change the type of a participating Feature after building a rule, Configurator Developer displays an error when you generate the Active Model.

A Property-based Compatibility rule:

- Enumerates the Features and BOM Option Classes that participate in the rule
- Specifies the Properties to be checked for compatible values
- Specifies the type of comparison to be made between the Property values, using standard numeric and string comparison operators, such as equals, greater than, contains, and matches

A rule can include multiple comparisons among the participating nodes.

5.10.1.1 Building Property-based Compatibility Rules

Before creating a Property-based Compatibility rule, be sure that:

- Each Feature that participates in the rule contains Options, and each Option has one or more Properties
- All of the Feature Options share at least one common Property

For example, to define a Property-based Compatibility rule involving Feature1 and Feature2, all of the Options must have at least one Property in common (although the Property *values* may be different). You can then select the Property that is shared by each Option when defining the rule. (Note that any Properties that are not common to *all* of the Options participating in the rule are not available when defining the rule.)

Example:

Feature1:

Option1 Property Name = Color, Value = Black

Option2 Property Name = Color, Value = White

Feature2:

Option1 Property Name = Color, Value = Red

Option2 Property Name = Color, Value = Green

For a general description of creating configuration rules, see Section 5.5 on page 5-20.

1. Select the arrowhead to the left of the **Definition** section to open it.
2. The **A** side of the rule consists of a comparison operator and two operands. Select one of the following operators from the list:
   - <> not equal
   - = equal
   - > greater than
   - **<** less than
   - < less than or equal to
   - > greater than or equal to
   - contains
   - does not contain
   - begins with
   - does not begin with
   - ends with
   - does not end with
   - matches

     See Section 5.10.1.1.1 on page 5-35 for more information about the equals (=), contains, and matches operators.

     Note: You can use any of these operators when comparing either numeric Properties or text Properties.

3. The left-hand operand is an Option Property. From the first list, select the Feature or BOM Option Class for which you want to compare a Property. From the list below it, select the Property you want to compare.
4. The right-hand operand is an Option Property or constant value. From the first list, select the Feature or BOM Option Class for which you want to compare a

Property, or (value). In the next field, type in a constant or select a Property from the list.

5. Select **And** or **Or**, and then follow the preceding steps to add another Property comparison to the rule.

Note: Configurator Developer evaluates Property-based Compatibility rules from the top down, and gives no priority or precedence to an expression based on its use of the AND or OR operator. In other words, the system evaluates the first relation you enter, followed by the second, and so on.

5.10.1.1.1 Equal, Contains, and Matches These operators are used when comparing Properties that consist of text, rather than numeric values. For example, a Property for the BOM Standard Item Copper Pipe is Diameter.

Properties may contain the percent sign (%) and underscore (_) characters. By using these characters, the equal (**=**), **contains**, and **matches** operators enable you to apply the Property-based Compatibility rule to only specific options. For example, when using the equal operator (=), the rule checks whether the Property on the **B** side of the rule contains the percent sign or underscore character itself.

When used with the **contains** or **matches** operators, the percent sign and underscore characters act as "wildcards." The percent sign can represent zero or more characters, while the underscore represents a single character. For example, "Dia%" will find a match for any word that begins with "Dia", while "_eight" will find a match for both "Height" and "Weight".

For example, the rule `AddressFeature.address contains "De"` is the same as `AddressFeature.address Matches %De%"`. Both rules find a match if the Property on the **A** side of the rule contains "De".

The equal operator does *not* use the percent sign or underscore characters as wildcards. So, the rule `AddressFeature.address = "Diam%"` does not find a match if the Property on the **B** side of the rule is Diameter; it finds a match only if the Property is "Diam%".

5.10.2 Explicit Compatibilities

Explicit Compatibility rules express compatibility constraints among Options of your Model that cannot be described in terms of a shared Property. An Explicit

Compatibility rule allows you to simply specify, in tabular form, explicit matches between the Options of one or more Features.

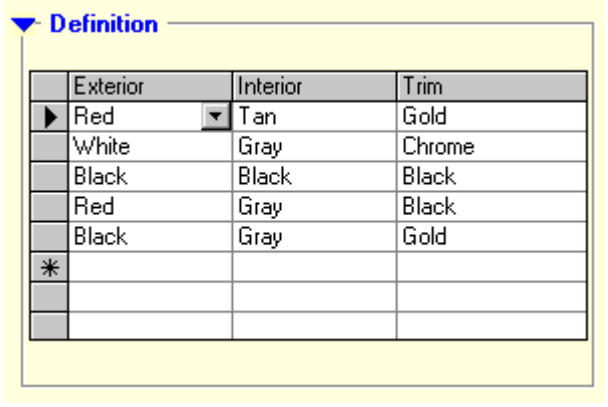
For example, consider an automobile model line with several color choices. Table 5–3 lists which colors (Options) are compatible with each Feature.

Table 5–3 Compatibility Rule Example

Features	Options		
Exterior	Red	White	Black
Interior	Tan	Gray	Black
Trim	Gold	Chrome	Black

Some color combinations are available, others are not. The available color combinations can be expressed in a compatibility table as shown in Figure 5–11.

Figure 5–11 Compatibility Rule Table



Each row in the table defines a valid combination of exterior, interior and trim colors. There can be no blank cells in the table. Options can be repeated in a column as many times as needed to define the available combinations. In this example, the Red exterior can have a Tan or Gray interior, but not Black. The Gray interior color goes with anything.

Features used in an Explicit Compatibility table must be of type List of Options. As described in Section 5.10, "Compatibility Rules" on page 5-31 the behavior of the

rules can be confusing if participating Features have Maximum Number of Selections greater than one.

5.10.2.1 Building Explicit Compatibility Rules

For a general description of creating configuration rules, see Section 5.5 on page 5-20.

To create an Explicit Compatibility rule:

1. Expand the **Definition** attribute.

   The table in this section contains a column for each of the Features you selected as Participants.

2. Use the cells of the column to specify Options of the Feature.

3. Select the cell to reveal the dropdown list button. Use this button to select an Option from the Options list for the Feature.

   There cannot be any blank cells in the Explicit Compatibilities table, so each column must have the same number of rows. The same Option can appear in the column as many times as necessary to build the compatibility table you need.

5.10.3 Gated Combinations

Gated Combinations refers to the conditions at runtime that cause Oracle Configurator to propagate False in Explicit Compatibility rules, Property-based Compatibility rules, and Design Charts. These conditions are determined by a setting in the CZ_DB_SETTINGS table called GenerateGatedCombo.

The default value of this setting is `Yes` and it produces runtime behavior that is considered preferable as it causes Oracle Configurator to exclude *more* incompatible selections than it would otherwise. However, if you recently upgraded your Oracle Configurator Developer installation and your Compatibility rules or Design Charts are producing different and undesirable results, you can restore the old behavior by setting `GenerateGatedCombo` to `No`. To do this, refer to the section about the CZ_DB_SETTINGS table in the *Oracle Configurator Implementation Guide*.

5.10.3.1 Behavior Using Gated Combinations

Compatibility rules are expected to propagate FALSE along a row of an Option in a Feature that is selected when all of the following are true:

- There is only one Feature in this Compatibility rule that is not selected

- The Option belongs to the remaining unselected Feature
- The Option is unavailable (FALSE)
- The Option does not belong to another row of the Compatibility table that is invalid

Or, all of the following are true:

- All of the Features of the Compatibility table have been selected
- The Option is unavailable (FALSE)
- The Option does not belong to another row of the Compatibility table that is invalid

If you recently upgraded your Configurator Developer installation, you may notice that the behavior of Compatibility rules has changed in the following circumstances:

1. The rule includes one or more participant Features that are optional (in other words, the Feature's **Minimum Number of Selections** is zero).

   Setting `GenerateGatedCombo` to `No` may force available selections to be False based on the possibly incorrect assumption that the user will eventually make a selection from an optional Feature. But a Compatibility rule should be interpreted to constrain only *complete sets of selections* from *all* of the participant Features; if an end user makes selections only from a subset of the participant Features, the Compatibility rule should not affect their selections. That is why this behavior, which is preferable, is the default as it causes *fewer* Options to be set to False than it would otherwise.

Example 5–10 One or More Features with a Minimum Number of Selections Equal to Zero

A Compatibility rule constrains selections from Features X (**Minimum** = 1 and **Maximum** = 1), Y (**Minimum** = 0, **Maximum** = 1), and Z (**Minimum** =0, **Maximum** = 1). Unless and until the end user makes a selection from Z, or a rule requires that they do so, the end user should be permitted to select Options X1 and Y1 even if no allowed combination of X, Y, and Z contains X1 and Y1.

2. The rule includes one or more participant Features with a **Maximum Number of Selections** greater than one.

   Setting `GenerateGatedCombo` to `No` prevents assumptions based on excluded Options from such Features until the maximum number of selections is reached. A Compatibility rule should be interpreted to require that every selected Option

must be part of a combination allowed by the rule. Any excluded (False) Option from such a **Maximum** > 1 Feature could be used to rule out Options from the other participants that are compatible only with the excluded Option, even if the maximum number of selections has not been reached.

The default behavior (GenerateGatedCombo set to `Yes`) permits this assumption and causes *more* Options to be forced False.

Example 5–11 One or More Participant Features with Maximum Number of Selections Greater than One

A Compatibility rule constrains selections from Features A (**Minimum** = 1 and **Maximum** = 1) and B (**Minimum** = 1 and **Maximum** = 2). The rule allows the following combinations of selections from A and B: {A1, B1}, {A1, B2} and {A2, B3}. In addition, a Logic rule states that Option X Excludes B3. If the user selects Option X, the Excludes rule causes B3 to become False. Because of the Compatibility rule, the system concludes that A2 must be False, because no other selection from B is compatible with A2.

In most Models, the default behavior provides more desirable behavior from the end user's point of view. It avoids invalid assumptions, and makes more valid ones. In a Compatibility rule, the end user cannot tell whether an Option is Unknown or False, as both False and Unknown Options are not visible in the UI.

If you upgraded from a previous release of Configurator Developer, the default behavior will affect the outcome of a configuration session only if your Model contains rules that depend on the distinction between Unknown and False. There are two types of rules that can do this:

1. Logic rules that use the Negates relation, or have Advanced Expression operands that contain the NOT operator. The consequences of these rules are propagated when the Negates or NOT operator's operand is False but *not* when it is Unknown. (See Example 5–12 on page 5-40.)

   Note: This is not the case for rules that use the NotTrue operator.

2. Numeric or Comparison rules involving Numeric Features or Totals with no initial values where the numeric result distinguishes between a value of zero and an Unknown value. (See Example 5–13 on page 5-40.)

Example 5–12 Propagation of False

A and B are Options. (NOT A) Implies B. In this case, B is True (selected) when A is False but *not* when A is Unknown.

Since it is difficult (and inadvisable) to force all unselected Options to be False, rules of this type are not recommended.

Example 5–13 Rule With No Initial Value

T is a total with no initial value. A and B are options. (A*1) Contributes To T. (T<1) Implies B. As in the example above, B is True (selected) when A is False but *not* when A is Unknown.

Rules like this are likely to produce unexpected results and should also be avoided.

5.11 Design Charts

A Design Chart is a way to express the complex explicit compatibility relationships that are often used in industry. It provides an easy way to define the relationship between a Model's *Primary* and *Secondary* Features.

A Primary Feature is a Feature with Options that define the variations of the product. The compatibilities defined in the Design Chart are based on this Feature. A Secondary Feature can be either a *Defining* or *Optional* Feature. A Defining Feature is a Feature with Options that participate in the unique combinations that define the options of the Primary Feature. An Optional Feature is a Feature whose Options can be arbitrarily compatible or incompatible with the Options of the Primary Feature.

For example, a hypothetical automobile company designs and manufactures several models of sport and full-size pickup trucks. Part of the Oracle Configurator Developer Model structure appears as shown in Figure 5–12.

For important information about the default runtime behavior of Design Charts, see Section 5.10.3, "Gated Combinations" on page 5-37.

Figure 5–12 Example of Automobile Model Structure

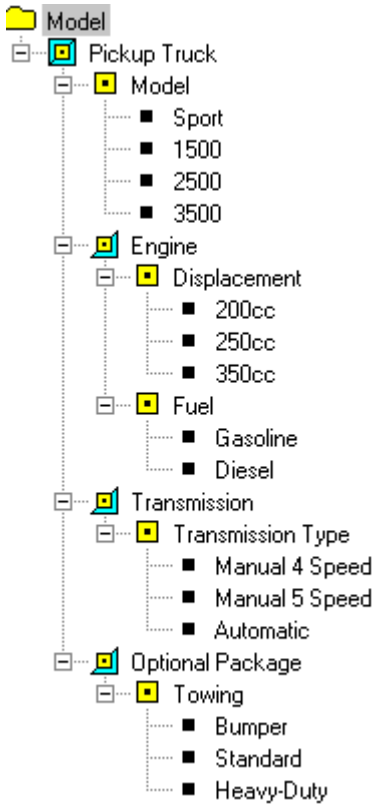
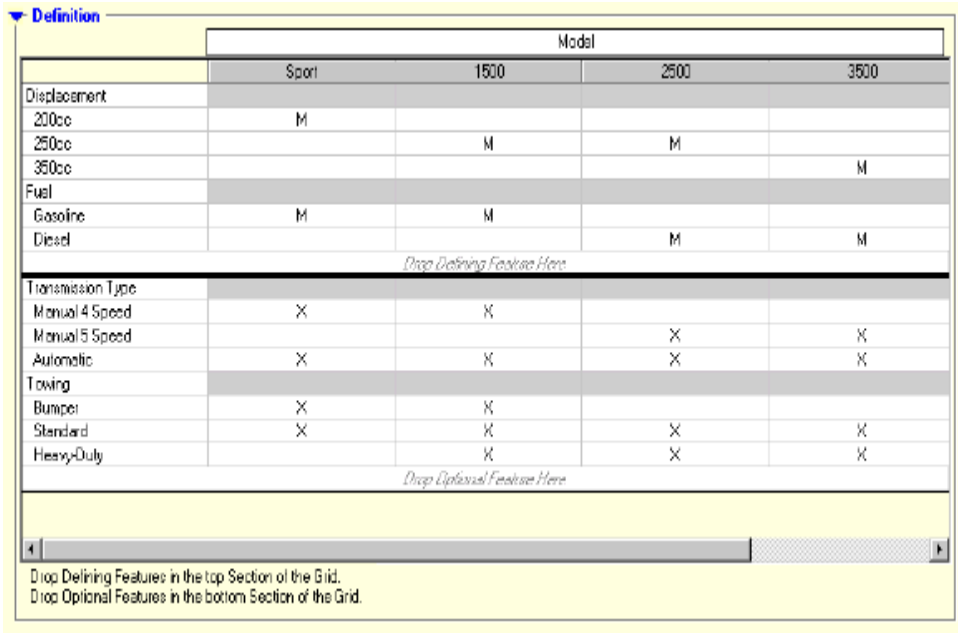


Figure 5–13 shows the Design Chart for this product.

Figure 5–13 Design Chart for Automobile Model Structure



Model is the Primary Feature. Its options represent the types of trucks that are available. Each truck model is defined by the engine displacement and fuel used. Therefore, **Displacement** and **Fuel** are the Defining Secondary Features. Each Option of the Primary Feature is defined by a unique combination of Defining Secondary Features.

Only one Defining Feature Option can be compatible with a given Primary Feature Option. When multiple Defining Feature Options are specified, as in this example, the combination of Options must be unique for each Primary Feature Option. Each column contains a unique combination of Options, therefore a row cannot contain more than one marked cell. Note how the Options specified for Displacement and Fuel are unique for each truck model.

Various combinations of transmission type and towing package are available for each truck model, so these become Optional Secondary Features. When the end user chooses a Sport model truck, it must have a 200cc gasoline engine, but it can have either a manual 4 speed or automatic transmission.

Features used in a Design Chart must be of type List of Options, and the **Number of Selections** for Primary Features and Defining Secondary Features must be **Minimum** = 0 or 1, and **Maximum** = 1.

Defining a Design Chart in Configurator Developer sets up a network of relationships in the runtime Oracle Configurator so that if the end user selects any of the Options included in the Design Chart, that selection can affect the logic state of other Options in the Design Chart.

Table 5–4 shows the effects at runtime when an end user selects the Sport model and the minimum **Number of Selections** on all Options is 0 (zero).

Table 5–4 Runtime Effects of Selecting Design Chart Options - Example 1

Feature	Option	Logic State
Model	Sport	**User True**
	1500	**Logic False**
	2500	**Logic False**
	3500	**Logic False**
Displacement	200cc	Available
	250cc	**Logic False**
	350cc	**Logic False**
Fuel	diesel	**Logic False**
	gasoline	Available
Transmission Type	Manual 4 Speed	Available
	Manual 5 Speed	**Logic False**
	Automatic	Available
Towing	Bumper	Available
	Standard	Available
	Heavy-Duty	**Logic False**

In this example, the end user selects a Sport model truck. This selection makes the logic state of the Option 'Sport' true by user selection. Because the maximum Number of Selections on this Feature is 1, the logic state of the other truck models becomes Logic False. Throughout the rest of the Model, those Options that are not

defined as compatible by the Design Chart become Logic False. Those Options that are defined as compatible remain Available. This scenario assumes the minimum Number of Selections on all Options is 0.

Table 5–5 shows the effects at runtime when an end user selects the Sport model and the minimum Number of Selections on all Options is 1.

Table 5–5 Runtime Effects of Selecting Design Chart Options - Example 2

Feature	Option	Logic State
Model	Sport	**User True**
	1500	**Logic False**
	2500	**Logic False**
	3500	**Logic False**
Displacement	200cc	Logic True
	250cc	**Logic False**
	350cc	**Logic False**
Fuel	diesel	**Logic False**
	gasoline	Logic True
Transmission Type	Manual 4 Speed	Available
	Manual 5 Speed	**Logic False**
	Automatic	Available
Towing	Bumper	Available
	Standard	Available
	Heavy-Duty	**Logic False**

In this case, if Design Chart constraints leave only one available Option, that option becomes Logic True.

Table 5–6 shows how selecting a Defining Secondary Feature constrains the Primary Feature. The end user selects a 250cc engine, which limits truck model choices to 1500 or 2500.

Table 5–6 Runtime Effects of Selecting Design Chart Options - Example 3

Feature	Option	Logic State
Model	Sport	**Logic False**
	1500	Available
	2500	Available
	3500	**Logic False**
Displacement	200cc	**Logic False**
	250cc	**User True**
	350cc	**Logic False**
Fuel	diesel	Available
	gasoline	Available
Transmission Type	Manual 4 Speed	Available
	Manual 5 Speed	Available
	Automatic	Available
Towing	Bumper	Available
	Standard	Available
	Heavy-Duty	Available

5.11.1 Building Design Charts

For a general description of creating configuration rules, see Section 5.5 on page 5-20.

1. Click on the arrowhead to the left of the **Definition** section to open it. The table in this section displays a column for each of the Primary Feature's Options. The rows of each column are where you select the compatible Defining or Optional Secondary Feature.
2. Select the Feature or BOM Option Class node from the Model View that is to be the **Primary Feature** and drag it to the table heading. The table is automatically populated with a column for each Option of the Primary Feature. A Primary Feature must be defined before the table is saved and you can activate the compatibility cells.
3. Select a Feature node from the Model View that is to be a **Defining Feature** and drag and drop it in the **Defining Feature** section of the table. A table row is

automatically populated for each of the Defining Feature's Options. There is no limitation on the number of Defining Features you can include. However, Defining Features should always have a **Number of Selections** of **Minimum** = 0 or 1, and **Maximum** = 1. This is because allowing the end user to choose more than one option from a Defining Feature can make it difficult to map choices to a particular Primary Feature selection.

4. Select a Feature node form the Model View that is to be an **Optional Feature** and drag and drop it in the **Optional Feature** section of the table. A table row is automatically populated for each of the Optional Feature's Options. There is no limitation on the number of Optional Features you can include.

   If you drag a node using the left mouse button, the Feature and its Options drop wherever you position the mouse in the table. If you drag a node using the right mouse button, a menu displays for you to select the destination of the Feature and its Options.

   Once you have populated the table with all of the Defining and Optional Features, select the table cell to select the **Defining Feature Option** that is compatible with each of the Primary Feature Options. A mark appears in the cell to indicate that you have selected it. The default mark for the Defining Feature Options is an M to indicate that they are mandatory. Continue selecting appropriate table cells for all compatible Options.

   You can customize the cell activation marks to be more representative of your business by modifying the `[Design Chart]` section of your `spx.ini` file. You must specify two different cell marks, one for the Defining Feature Options and one for the Optional Feature Options. You can enter one or more cell marks for each option.

   For example, if you want to set A, B, and C for the Defining Feature Options and Y and Z for the Optional Feature Options, your spx.ini file would look like this:

   ```
   [Design Chart]
   DEF=A, B, C
   SEC=Y, Z
   ```

   If you enter multiple values, Configurator Developer cycles through the values each time you click in the same Design Chart cell.

   See the *Oracle Configurator Implementation Guide* for more information on the `spx.ini` file.

5. Select the table cell to select the **Optional Feature Option** that is compatible with each of the Primary Feature Options. A mark is placed in the cell to

indicate it has been selected. The default mark for the Optional Feature Options is an X. Continue this for all compatible Options. An Optional Feature Option can be compatible with multiple Primary Feature Options and multiple Options of a given Optional Feature can be compatible with a given Primary Feature Option.

If you select a cell and want to deselect it, you can do so by using the mouse, or positioning the cursor in the cell and pressing the **Spacebar**, **Delete** key, or **Backspace** key.

If you want to remove a Secondary Feature (Defining or Optional) from the table, position the cursor on the Feature's name and press **Delete** or use the right mouse button to select **Delete** from the menu. You *cannot* delete Options from the Design Chart. Note that deleting a Feature also deletes all of its Options.

5.12 Rule Sequences

A Rule Sequence is an ordered set of rules whose effective dates are ordered sequentially, with each rule in the sequence becoming active as soon as its predecessor becomes inactive. You may want to create a Rule Sequence if, for example, you want to automatically deactivate a rule and activate a new one based on a specific date and time that you specify.

You define effective dates to each rule in a Rule Sequence to determine when each rule in the set becomes active, how long it is used, and when it becomes inactive. Since a rule must become active as soon as its predecessor's becomes inactive, modifying the effective date of one rule in a Rule Sequence can affect the effective dates of at least one other rule in the set. Each rule in a Rule Sequence can either have an effective date range defined or be assigned to an Effectivity Set. See Section 5.12.2, "Rule Sequences and Effectivity Sets" on page 5-48.

5.12.1 Viewing Rule Sequences

Rule Sequences appear in the Configuration Rules view in the Rules module and are represented by a unique node in the tree. Rules that belong to the Rule Sequence are presented as children of the Rule Sequence node in the order in which they appear in the sequence. This order also represents the chronological order in which each rule in the set becomes active over time.

Rules in a Rule Sequence appears only as children of the Rule Sequence to which they belong, they do not appear elsewhere in the Rules view (for example, in the

Logic Rules or Numeric Rules folder). If you remove the rule from a Rule Sequence later, it appears in its original folder in the Rules view.

Additionally:

- When viewing rules by folder, the Rule Sequence is displayed as a child of the Rules Folder that contains it. (To create a rule folder, you must first choose **View > Rules > By Folders**.)
- When viewing rules by name, the Rule Sequence itself is alphabetized under the Configuration Rules root folder.
- When viewing rule by type, Configurator Developer displays a separate Rule Sequences folder.

When you select a Rule Sequence in the Rules tree, the Attributes view displays its Name, Description, and all rules that it contains. You can then select individual rules in the sequence and remove or reorder them, or click a button to enable or disable all rules in the Rule Sequence. To view or edit a rule's definition, select it in the Rules tree, then expand the **Definition** attribute.

5.12.2 Rule Sequences and Effectivity Sets

When you assign start and end dates to an Effectivity Set or alter its start and end dates, and the Effectivity Set contains members of one or more Rule Sequences, Configurator Developer displays all of the error conditions associated with Rule Sequence date constraints and rejects the new date(s). If there are no errors, Configurator Developer applies the new dates to each Rule Sequence that contains a rule associated with the Effectivity Set.

Sometimes an operation on a Rule Sequence requires a change in the effective dates of a rule that conflicts with the effective dates of the Effectivity Set associated with the rule. In these cases, the rule whose effective dates need to be changed will be disassociated from its Effectivity Set and the new effectivity dates will be applied directly to the rule itself. When this occurs, Configurator Developer displays a warning message and you can choose to cancel or continue with the operation. Some examples of this situation include:

- Activating the first inactive rule in a Rule Sequence when its predecessor is a member of an Effectivity Set.
- Adding a rule to a Rule Sequence in a position such that its new predecessor and / or successor is active and is a member of an Effectivity Set.
- Removing a rule from a Rule Sequence if its predecessor and / or successor is active and is a member of an Effectivity Set.

- Moving a rule within a Rule Sequence such that either of the previous two conditions apply.
- Adding a rule to a Rule Sequence or moving a rule within a Rule Sequence, if the rule being added or moved is itself a member of an Effectivity Set and its new position in the Rule Sequence requires a change to its own effective dates.

5.12.3 Creating a Rule Sequence

When you create a new Rule Sequence, it contains no rules. You must specify each rule that you want to add to the sequence. If the rule you add is the only member of the sequence, its default effective date range is Always Effective. Otherwise, the new rule's effective date range depends on the end date of its predecessor in the sequence.

The effective dates of rules in a Rule Sequence cannot overlap. When you add a rule that is effective at some point in time to a Rule Sequence, Configurator Developer places the new rule at the end of the sequence and changes its start date to its predecessor's end date. If the rule you are adding is set to Always Effective, Configurator Developer displays a message that the rule's effectivity will be changed to Never Effective. You must then modify its effective date as required.

All rule types except Functional Companions can participate in a Rule Sequence. However, a Rule Sequence cannot contain another Rule Sequence.

To create a Rule Sequence:

1. Go to the Configuration Rules module, then:
2. Select **Create > New Rule Sequence** or right click and choose **New Rule Sequence**. The system creates a new Rule Sequence node in the *Model Name Rules* folder.
3. Rename the Rule Sequence (optional).
4. Add rules to the Rule Sequence using menu commands or by dragging and dropping using the mouse. For example:
   a. Select a rule.
   b. Choose **Edit > Add to Sequence**, or right click and select **Add to Sequence**.

c. Select the Rule Sequence from the Add to Sequence dialog. Alternatively, right-click on the rule and simply drag and drop it onto the Rule Sequence.

Note that Configurator Developer places each rule you add at the end of the sequence and, if necessary, automatically updates its effective dates. For more information, see Section 5.12.4, "Modifying a Rule Sequence" on page 5-50.

5.12.4 Modifying a Rule Sequence

When you change the effective date range of a rule in a Rule Sequence, Configurator Developer adjusts the date ranges of the rules that precede and follow that rule (if any) to maintain the constraints inherent in the sequence. If this is not possible, the system displays an error. If necessary, you can change a rule's effective dates or modify the Effectivity Set to which it is assigned after adding it to a Rule Sequence.

Following are a few examples of what occurs when you modify effective dates for rules in a Rule Sequence:

- You assign a start date to the first inactive rule in the sequence, or modify the start date of an active rule. Configurator Developer adjusts the end date of the preceding rule (if any) to match. If it cannot do this without changing the dates of other rules in the sequence, Configurator Developer displays an error message and rejects the new date.
- You modify the end date of an active rule, and the following rule is active, Configurator Developer adjusts the start date of the rule that follows. If it cannot do this without changing the dates of other rules in the sequence, Configurator Developer displays an error message and rejects the new date.
- You deactivate the last active rule in a sequence by setting it to Never Effective, and that rule has a predecessor. Configurator Developer sets the end date of the predecessor to what was previously the deactivated rule's end date.
- You deactivate a rule that is not the last active rule in the sequence. Configurator Developer displays an error message. To deactivate a rule, you must move it after the last active rule in the sequence, and then set it to Never Effective.
- You assign start or end dates to an inactive rule whose predecessor is not active. Configurator Developer displays an error message because the first rule that is active must appear first in the Rule Sequence.

5.12.4.1 Reordering Rules in a Rule Sequence

When you change the order of rules in a Rule Sequence, Configurator Developer displays a message that tells you how the effective dates for each rule will be affected. Then, depending on whether the new dates are acceptable, you can either proceed or cancel the operation. If you click **OK**, Configurator Developer changes the effective date ranges the effective dates of the rules. If you click **Cancel**, the sequence of rules and their effective dates are not changed.

To change the order of rules in a Rule Sequence:

1. Select the Rule Sequence, then expand the **Rules** attribute.
2. Select the rule you want to reorder in the sequence, then click **Reorder**.
3. In the Reorder Rule dialog, click the up or down arrow to modify the location of the rule in the sequence.

   Note that you can modify only the selected rule. To reorder another rule in the sequence, you must close the Reorder Rules window, select the rule to modify, then click **Reorder**.

Alternate method:

1. Expand the Rule Sequence, then select the rule you want to move.
2. Choose **Edit > Reorder in Sequence**, or right click using the mouse and select Reorder in Sequence.
3. In the Reorder Rule dialog, click the up or down arrow to modify the location of the rule in the sequence.

5.12.4.2 Reordering Rules and Rule Effective Dates

Following are some examples of how effective dates may change when you reorder rules in a Rule Sequence:

- If the rule that is moved is inactive (that is, never effective and appearing at the end of the sequence), its start and end date are adjusted to be consistent with its new position.
  - If its new position is at the end of the sequence or its new successor is also inactive, no changes in effectivity are necessary.
  - If its new successor's start date is unbounded, the rule's start date is made equal to its new successor's old start date, the rule's end date is made equal to its new successor's old end date, and the new successor's start and end dates are both made equal to its old end date.

- If its new successor's start date is finite, the rule's start date and end date are both made equal to the new successor's start date.

- If the rule that is moved is active (effective for some period of time), Configurator Developer:
  - Adjusts the start and end dates of the rule's old predecessor's and / or successor's to fill the gap left when you moved the rule. You can then modify these dates as required.
  - Adjusts the start and end dates of the rule that you moved to be consistent with its new context. For example:
    * If its new predecessor is inactive, the rule that you moved becomes inactive.
    * If its new successor's effective dates range have specific start and end dates, both the start and end date of the rule that you moved are set to its predecessor's end date. You can then modify these dates as required.

5.12.4.3 Removing Rules from a Rule Sequence

You can remove rules from a Rule Sequence at any time by using cut and paste or by clicking a button. When you remove a rule, its effective dates do not change. However, Configurator Developer adjusts the effective dates of the remaining rules in the Rule Sequence so no gaps exist between them.

When you cut a rule from a Rule Sequence, it still appears as part of the Rule Sequence until you paste it into another location (for example, into a rule folder). When you remove a rule by clicking the **Remove** button, it appears in its original folder in the Rules view.

When you remove a rule from a Rule Sequence, Configurator Developer modifies the effective date ranges of the adjacent rules as required to maintain the constraints in the sequence. For example:

- If the deleted rule and its predecessor are active, the predecessor's end date is set to the deleted rule's end date.
- If the rule is inactive, the effective dates do not change.

To remove a rule from a Rule Sequence:

1. In the Rules module, select the rule.
2. Right click and choose either **Remove from sequence** or **Cut**.

3. If you chose Cut, select another location in the Rules view (such as a folder), then:
   - Right click and choose **Paste**, or
   - Choose **Edit > Paste**

Alternate method:

1. In the Configuration Rules module, select the Rule Sequence.
2. Expand the Rules attribute, then select the rule from the list.
3. Click **Remove**.

5.12.4.4 Enabling and Disabling Rules in a Rule Sequence

When unit testing a Model, you may want to enable or disable specific rules in a Rule Sequence to view the affect at runtime. You can also disable or enable *all* rules in a Rule Sequence simultaneously, depending on your needs. Any rules that are disabled are ignored at runtime.

Before viewing the affect of your changes in a configuration session, be sure to regenerate the Active Model (**Test > Generate Active Model**).

To enable or disable only one rule in a Rule Sequence:

1. In the Configuration Rules module, select the rule.
2. Expand the **Rules** attribute.
3. Select or deselect the **Disable** check box as appropriate.

To enable or disable *all* rules in a Rule Sequence:

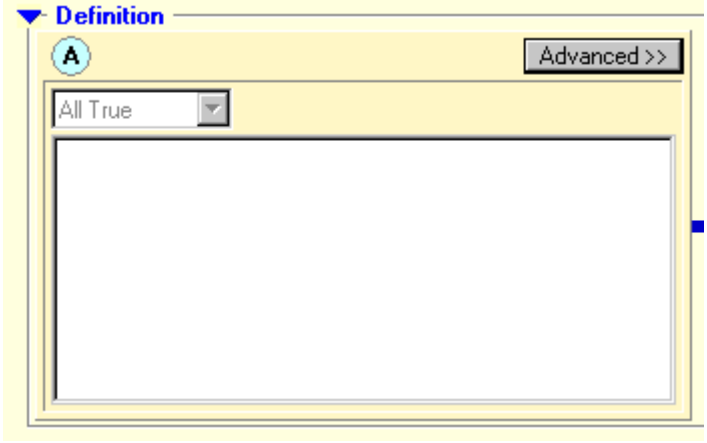
1. In the Configuration Rules module, select the rule Sequence.
2. Click **Enable all rules in the sequence**, or **Disable all rules in the sequence.**

5.13 Advanced Expressions

Advanced expressions enable you to incorporate complex expressions into your configuration rules. You can define advanced expressions for all rules except Explicit Compatibility rules, Functional Companions, and Design Charts.

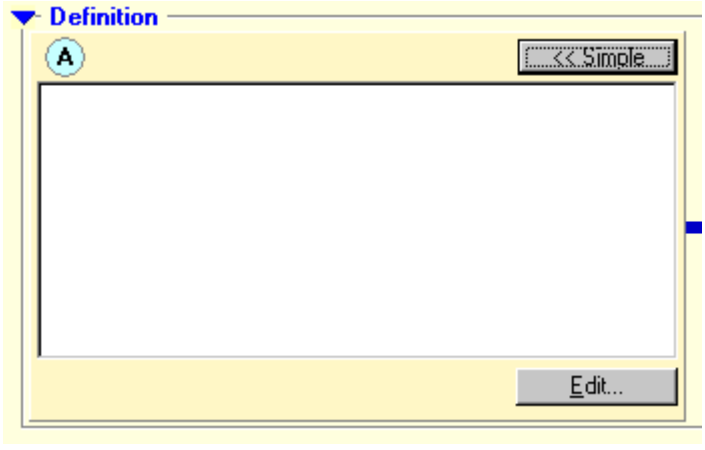
You build advanced expressions in the Advanced Expression editor, which you access by selecting the **Advanced** button in the **Definition** section of the Attributes View (see Figure 5–14).

Figure 5–14 The Advanced Expression Editor Button



Click **Edit** to open the Advanced Expression editor (see Figure 5–15 on page 5-54 and Figure 5–16 on page 5-55). Or, click **Simple** to create a simple expression.

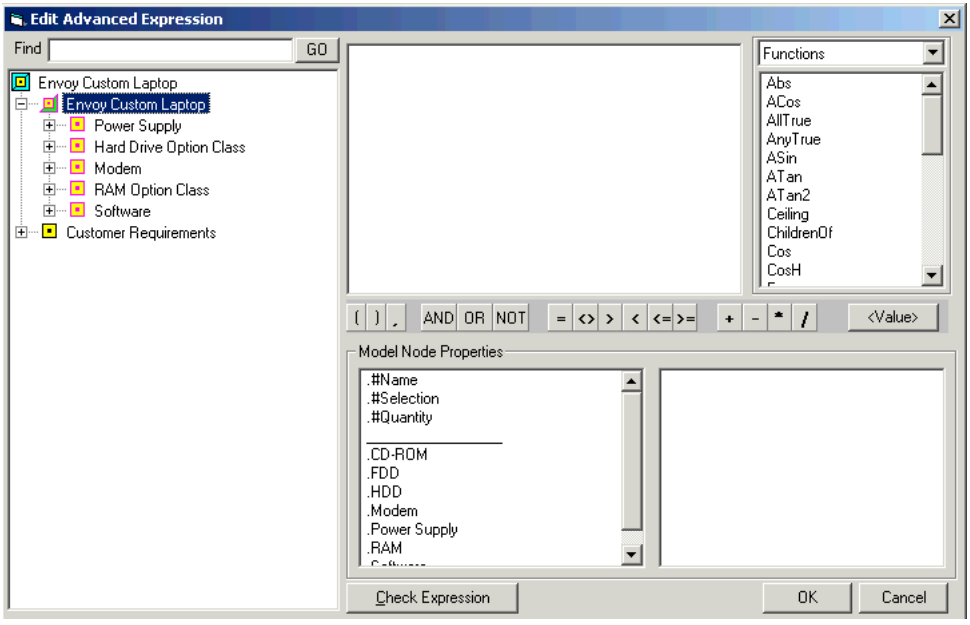
Figure 5–15 The Simple and Edit Buttons



5.13.1 The Advanced Expression Editor

The Advanced Expression editor provides a drag-and-drop interface for building configuration rules that use advanced expressions (see Figure 5–16 on page 5-55).

Figure 5–16 The Advanced Expression Editor



The Advanced Expression editor is divided into the following regions:

- The Model pane in the left portion of the window.
- The Expression pane in the upper center area of the window.
- The Operators pane to the right of the Expression pane.
- The **Model Node Properties** area in the bottom of the window.

You build advanced expressions in the Expression pane by dragging expression building blocks from other panes of the editing window and dropping them into the Expression pane. If you need to type a specific value into the expression, you must select the **<Value>** button, which opens a small input field in the Expression pane. You can also select and delete text from the Expression pane.

The Operators pane displays **Functions**, **Operators**, **Constants** or **Other** grouping and separator characters, depending on your selection from the list. The operators and other characters are also displayed as buttons immediately below the Expression pane. Click one of these buttons to add the character to the expression. For a more information, see:

- Section 5.13.2.1, "Operators" on page 5-58
- Section 5.13.2.4, "Functions" on page 5-59

5.13.1.1 Model Node Properties in the Advanced Expression Editor

The Model pane and the Model Node Properties area together display the Model structure and Properties of nodes in the Model. The Model structure appears in the left-hand pane. Properties for the selected Model structure node appear in the **Model Node Properties** pane.

System Properties display above a dividing line in the **Model Node Properties** pane and appear with a pound sign (#) to differentiate them from User Properties. User Properties display below the dividing line (see Figure 5–18 on page 5-57).

In Figure 5–17 on page 5-56, the selected node has no User Properties, but it does have the System Properties Name, Selection, and Quantity.

Figure 5–17 Model Structure and Node Properties

In Figure 5–18, the selected node has User Properties Diameter, Length, Size, and Type, and the System Properties Name and Quantity.

Figure 5–18 System and User Properties on a Model Node

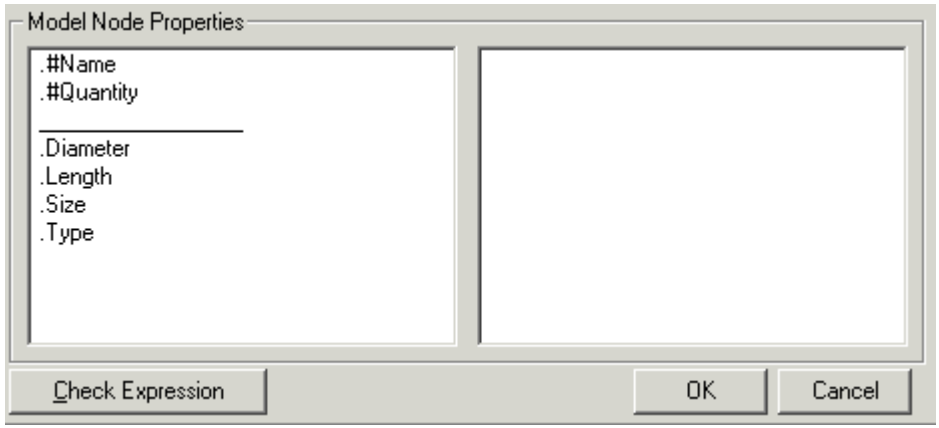
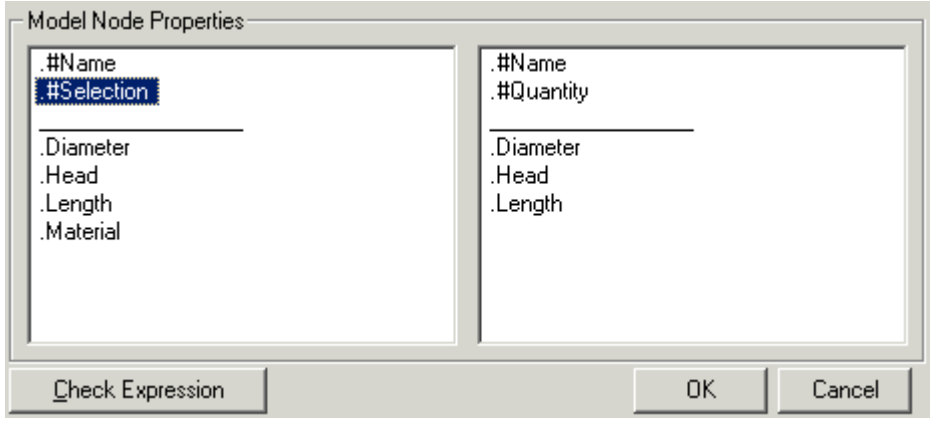


Figure 5–19 shows the System and User Properties for a List of Options Feature with one required selection (maximum = 1 and minimum = 1). This type of Feature has a System Property called **Selection**. When you select that Property, the lower-right screen area displays all Properties of the selected Feature's Options.

Figure 5–19 System and User Properties on a Feature Node



When the advanced expression is evaluated at runtime, the Selection Property evaluates to the Property of the Option selected by the Oracle Configurator end user.

For more information about Properties, see Section 3.2 on page 3-2.

5.13.2 Building Advanced Expressions

Advanced expressions are made up of operators and their operands, and functions and their arguments.

5.13.2.1 Operators

Table 5–7 lists the Operators you can use when creating an advanced expression.

Table 5–7 Advanced Expression Operators

Operator Type	Operators	Description
Logical	AND OR NOT	AND requires two operands and returns true if both are true. OR requires two operands and returns true if either is true. NOT requires one operand and returns its opposite value: false if the operand is true, true if the operand is false.
Arithmetic	* / - + [/]	Perform ordinary arithmetic operations on numeric operands.
Comparison	<> = > < < >	Perform the comparison operations of not equal (<>), equal (=), less than (<), greater than (>), less than or equal (<), greater than or equal (>). Possible comparisons include comparisons between a numeric valued Feature and a property of a selected Option, and comparisons between a Property value and the name of an Option.
Other	( ) , . -	parentheses ( ) are used to group sub-expressions comma (,) is used to separate function arguments dot (.) is used for referencing objects in the Model tree structure unary minus (-) is used to make positive values negative and negative values positive.

5.13.2.2 Precedence of Operators

Operators are processed in the order given in the following list. Operators with equal precedence are evaluated left to right.

1. Functions, operations within parenthesis ( ) and the dot (.) operator (for example, Function.Selection)
2. Unary minus (-10)
3. Multiplication (*) and division (/ or [/])
4. Addition (+) and subtraction (-)
5. Logical NOT
6. Equal to (=) and not equal to (<>)
7. Greater than (>), less than (<), greater than or equal to (>), and less than or equal to (<)
8. Logical AND, and OR

5.13.2.3 Operands

There are three types of Operands: logical (true or false), numeric, or complex. Logical Operands are sometimes referred to as Boolean expressions. Operands can be Component, Feature, or Option nodes from your Model, or they can be literal values entered after clicking the **<Value>** button.

Logical Operands can be Features, Options, or literal values.

Numeric Operands can be Option counts, Totals, Resources, integer, or decimal Features, or constant values.

Complex Operands can be Components, or Features with Options.

5.13.2.4 Functions

Functions perform operations on their arguments and return values which are used in evaluating the entire advanced expression. Functions must have their arguments enclosed in parentheses and separated by commas if there is more than one argument. Function arguments can be expressions.

For example, the following both have the correct syntax for the Round function, provided that Feature-1 and Feature-2 are numeric Features:

```
Round (13.4)
Round (Feature-1 / Feature-2)
```

Table 5–8 lists the logical functions that are available in Configurator Developer.

Table 5–8 Logical Functions

Function	Description
AllTrue	A logical AND expression. Accepts one or more logical values or expressions. Returns true if all of the arguments are true, or false if any argument is false. Otherwise, the value of AllTrue is unknown. See Section 5.1.6, "All True and Any True" on page 5-11.
AnyTrue	A logical OR expression. Accepts one or more logical values or expressions. Returns true if any of the arguments are true, or false if all arguments are false. Otherwise, the value of AnyTrue is unknown. See Section 5.1.6, "All True and Any True" on page 5-11.
NotTrue	Accepts a single logical value or expression. Returns True if the argument is False or unknown. If the argument is True, the value of NotTrue is unknown. See Section 5.13.3, "Using NotTrue in Advanced Expressions" on page 5-66.

Table 5–9 lists the arithmetic functions that are available in Configurator Developer. The term infinity is defined as a number without bounds. It can be either positive or negative.

Table 5–9 Arithmetic Functions

Function	Description
Abs(x)	Takes a single number as an argument and returns the positive value (0 to +infinity). The domain range is -infinity to +infinity. Returns the positive value of x. Abs(-12345.6) results in 12345.6
ACos(x)	Takes a single number between -1 and +1 and returns a value between 0 and pi. ACos(x) returns the arc cosine of x. An input outside the range between -1 and +1 results in an error.
ASin(x)	Takes a single number between -1 and +1 and returns a value between -pi/2 and +pi/2. ASin(x) returns the arc sine of x. An input outside the range between -1 and +1 results in an error.
ATan(x)	Takes a single number between -infinity and +infinity and returns a value between -pi/2 and +pi/2. ATan(x) returns the arc tangent of x.

Table 5–9 (Cont.) Arithmetic Functions

Function	Description
ATan2(x,y)	The arc tangent function is a binary function. The x and y values are between -infinity and +infinity. It returns a value between -pi and +pi. This is the four-quadrant tangent inverse.
Ceiling(x)	Takes a single decimal number as an argument and returns the next higher integer.
Cos(x)	Takes a single number between -infinity and +infinity and returns a value between -1 and +1. Returns the cosine of x.
Cosh(x)	Takes a single number between -infinity and +infinity and returns a value between -infinity and +infinity. Returns the hyperbolic cosine of x in radians. An error is returned if x exceeds the max of a double: cosh(-200) is valid whereas cosh(-2000) results in an error.
x / y	Division x / y propagates an error message when the divisor is 0.
Exp(x)	Returns e raised to the x power. Takes a single number between -infinity and +infinity and returns a value between 0 and +infinity.
Floor(x)	Takes a single decimal number as an argument and returns the next lower integer.
Log(x)	Takes a single number greater than 0 and less than +infinity and returns a number between -infinity and +infinity. Returns the logarithmic value of x. An error occurs if x=0.
Log10(x)	Takes a single number greater than 0 and less than +infinity and returns a number between -infinity and +infinity. Returns the base 10 logarithm of x. An error occurs if x=0.
Min(x,y,z...)	Returns the smallest of its numeric arguments.
Max(x,y,z...)	Returns the largest of its numeric arguments.
Mod(x,y)	This is a binary function. Returns the remainder of x / y where x and y are numbers between -infinity and +infinity. If y is 0, then division by 0 is treated as an error. If x=y, then the result is 0. For example, Mod(7,5) returns 2.
Pow(x,y)	This is a binary function. Returns the result of x raised to the power of y. The numbers x and y are between -infinity and +infinity and the returned result is between -infinity and +infinity. If y=0, then the result is 1. For example, Pow(6,2) returns 36.

Table 5–9 (Cont.) Arithmetic Functions

Function	Description
Round(x)	Takes a single decimal number as an argument and returns the nearest higher or lower integer. (If the B side of a Numeric rule contains an imported BOM item that allows decimal quantities, this function is not available. See Section 3.4.3.3, "Decimal Quantities" on page 3-11.)
RoundDownToNearest(x,y)	This is a binary function. x is a number between -infinity and +infinity, y is a number greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. The first argument is rounded to the nearest smaller multiple of the second argument. For example, RoundDownToNearest(433,75) returns 375.
RoundToNearest(x,y)	This is a binary function. x is a number between -infinity and +infinity, y is a number greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. RoundToNearest(433,10) returns 430.
RoundUpToNearest(x,y)	This is a binary function. The number x is between -infinity and +infinity, and the number y is greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. The first argument is rounded up to the nearest multiple of the second argument. For example, RoundUpToNearest(34.1,0.125) returns 34.125.
Sin(x)	Takes a single number x between -infinity and +infinity and returns a value between -1 and +1.
Sinh(x)	Returns the hyperbolic sine of x in radians. Takes a single number between -infinity and +infinity and returns a value between -1 and +infinity. An error is returned when the result exceeds the double. For example, sinh(-99) is valid but sinh(999) results in an error.
Sqrt(x)	Sqrt(x) returns the square root of x. Takes a single number between 0 and +infinity and returns a value between 0 and +infinity. An input of -x results in an error.
Tan(x)	Takes a single number x between -infinity and +infinity and returns a value between -infinity and +infinity.
Tanh(x)	Returns the hyperbolic tangent of x. Takes a single number x between -infinity and +infinity and returns a value between -1 and +1.

5.13.2.4.1 Arithmetic Function Overflows and Underflows

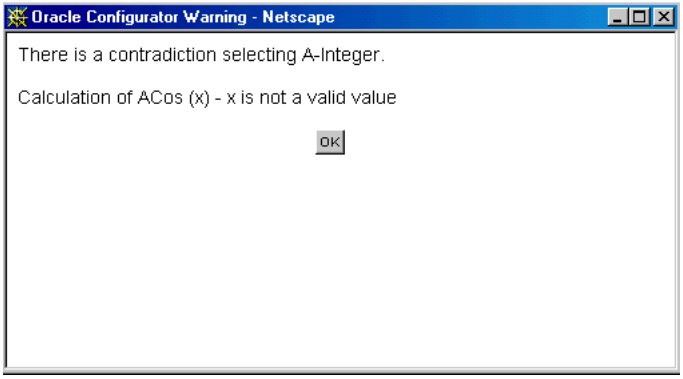
It is possible that some arithmetic functions produce an error either because of the resulting size (larger than the largest positive or negative double) or an invalid input. Entering a meaningful rule violation message can be helpful when debugging errors.

Following are some examples of possible error messages.

Example 5–14 Invalid Input Range Error

Following is a Numeric rule in which Acos(A-integer) contributes to a Total. When the input is out of the valid domain range (-1 to 1), an error message is returned. If the input is 2, the error shown in Figure 5–20 appears.

Figure 5–20 Invalid Input Range Error



Example 5–15 Intermediate Value Propagation Error

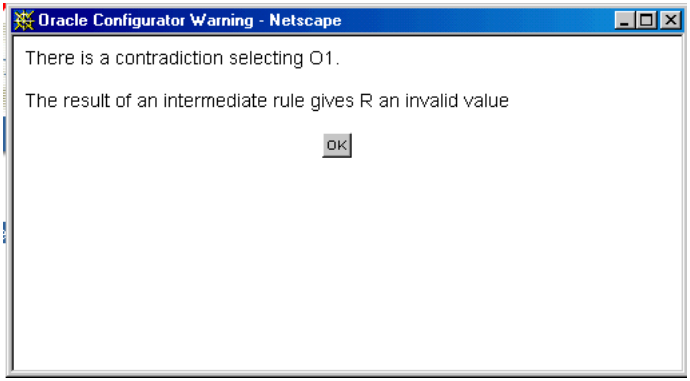
It is possible that propagation through some math functions results in an unexpected error because of an intermediate value propagated to the argument of the function. The following Model has a Feature with two counted Options (O1 and O2), a Resource (R) with no initial value (default is 0), and a Total (T) with no initial value (default is 0).

```
Numeric Rule 1: O1 *1 consumes from R

Numeric Rule 2: Sqrt(R) contributes to T
```

If O1 is 1, then R has a value of -1. Numeric rule 2 tried to calculate the Sqrt(-1) and returns the message shown in Figure 5–21.

Figure 5–21 Invalid Value Contradiction Message



Example 5–16 Calculated Input Value Out of Range Error

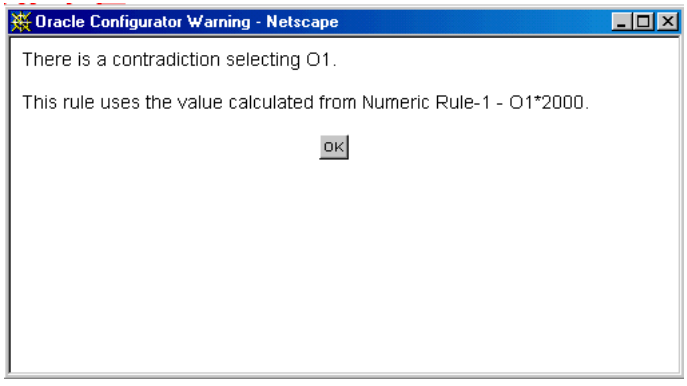
The following Model has a Boolean Feature (B1), a Feature with two counted Options (O1 and O2), a Resource (R) with no initial value (default is 0), and a Total (T) with no initial value (default is 0).

```
Numeric Rule 1: (O1)*2000 contributes to T

Numeric Rule 2: CosH(T) consumes from R
```

If `O1` is 1, then `T` has a value of 2000 and `CosH(T)` produces a result that is greater than the max of a double and returns the message shown in Figure 5–22.

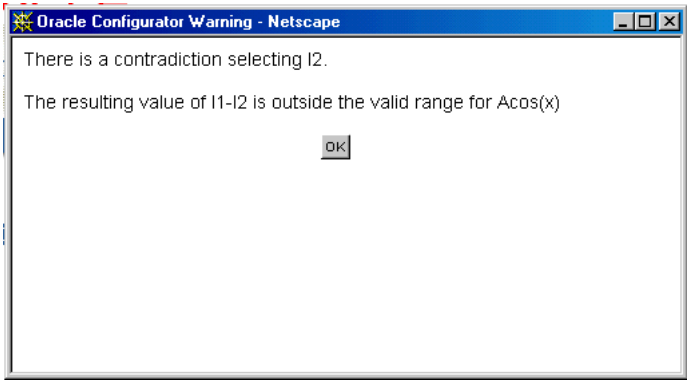
Figure 5–22 Contradiction Message



Example 5–17 Calculated Value Not Within Valid Range Error

A Model has two integer Features (I1 and I2) with initial values of 0. Totals (T1 and T2) with no specified initial values. The Numeric rule ACos(I1-I2) contributes to T1. If I1 is 1 and I2 is 3, then I1-I2 is outside the valid range (-1 to 1) for ACos(x). This produces the message shown in Figure 5–23.

Figure 5–23 Outside Valid Range Contradiction Message



Note: This behavior depends on the order in which the relations are propagated.

Table 5–10 lists the compound functions that are available in Configurator Developer.

Table 5–10 Compound Functions

Function	Description
OptionsOf	Takes a Feature as an argument and returns its Options.
FeaturesOf	Takes a Component as an argument and returns its Features.
ChildrenOf	Takes a BOM Model, BOM Option Class, or BOM Standard Item as an argument and returns its children.

5.13.3 Using NotTrue in Advanced Expressions

You can use NotTrue in an advanced expression to produce a logical consequence if some logical term is either false or Unknown. For example, you may want option Y to be selected whenever option X is not selected. If X is selected, Y may or may not be selected. You can do this by writing an Implies rule with NotTrue(X) on the left-hand side and Y on the right-hand side. For example:

NotTrue(X) Implies Y

This is different from the behavior of the Not function. If you replace NotTrue with Not in the above example, the rule would force Y to be True only when X is False. In that case, if X is Unknown, Y could also be Unknown.

Note that a rule with a NotTrue expression cannot "push back" on the argument to the NotTrue function. In other words, in the above example the end user could not cause X to become True by deselecting Y. The only way the user can remove Y from the configuration is to select X. Because of this, you should use NotTrue sparingly. Most logical values in a configuration start out Unknown, and using NotTrue can require an end user to make selections in a specific order.

Note also that logical cycles involving NotTrue can create Models where some items are "locked" in their initial states. For example, there are three options, A, B and C, with the following rules:

NotTrue(A) Implies B

B Excludes C

C Requires A

Option A starts out Unknown, which causes B to be True. The Excludes rule causes C to be False, and the Requires rule causes A to be False. The Configurator will not allow you to change any of these states. To prevent this behavior, avoid closing loops of rules involving NotTrue.

5.13.4 Advanced Expression Errors

You can check if your advanced expression is structured correctly by selecting the **Check Expression** button.

You may receive one of the following types of errors:

- Type Mismatch: The advanced expression does not evaluate to a datatype allowed in the portion of the rule you are building. For example, an expression that evaluates to a number is not valid in a logic rule.

- Type Mismatch: The argument of a function is the wrong data type. For example, `AllTrue(10)`.
- Type Mismatch: The operand of an operator is the wrong data type. For example, `1 AND 2`.
- Expected: Operand: The expression does not contain the correct number of operands. For example, the following expression is missing an operand:

  ```
  3 +
  ```

- Expected: Operator: Your expression does not contain the correct number of operators. For example "3 3" is invalid because there is no Operator between the two numbers. However, "3 * 3" is a valid statement.

5.13.5 Building Advanced Expressions for Rules

1. Click the **Advanced** button.
2. Click the **Edit** button to open the **Advanced Expression** editor (see Section 5.13.1, "The Advanced Expression Editor" on page 5-55).
3. Drag and drop the specific operand node(s) from the Model tree to the rule expression. You can also use node Properties as rule participants.

   To search for a node in the Model structure, enter a string in the **Find** field, then click **Go**. The search highlights the first node in the structure containing the string you entered. Click **Go** again to find other nodes matching the search string. (The **Find** field is not case sensitive.)
4. Select the corresponding arithmetic, comparison, or logical Operator(s) buttons to add to the expression wherever you place the cursor. Select the **Value** button to add a numeric literal to the expression.
5. Click **Check Expression** to validate the expression.
6. Click **OK**.
7. Click **Generate Active Model** from the **Tools** menu.
8. Test your new rule (see Section 7, "Testing and Debugging" on page 7-1).

5.14 Functional Companions

Functional Companions extend your runtime Oracle Configurator by attaching custom code through established interfaces.

To enable your Functional Companion to work with your configuration model, you must associate it with a node in your Model. You create this association in Oracle Configurator Developer, as a type of configuration rule.

Functional Companions on your Model work in any runtime Oracle Configurator. In the Java applet runtime Configurator, they must function without any end user action through the User Interface, because there are no visible buttons to launch Functional Companions. However, a DHTML UI launched from the Java applet can use visible buttons to launch Functional Companions. See Section 6.1.2.1, "Runtime Oracle Configurator: Java Applet" on page 6-3.

For information on building Functional Companions, see the *Oracle Configuration Interface Object (CIO) Developer's Guide.*

5.15 Rules that Relate Components and Models

There are some restrictions you must keep in mind when constructing rules that relate Models, Components, and References (this section refers to these nodes collectively as "components"). A rule **relates** two components if one or more nodes from each component's structure are participants in the rule.

To understand the restrictions that exist when relating optional components, you must understand following terms that have specific meaning within Oracle Configurator Developer:

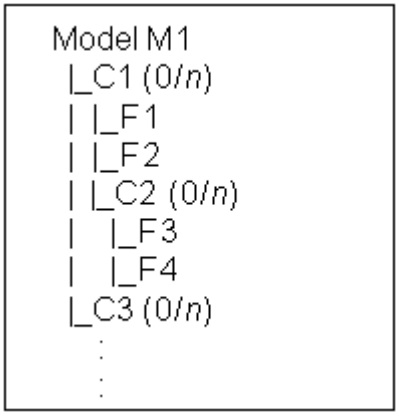
- A **required component** has a minimum number of **Instances** of one, a maximum number of **Instances** of one, *and* no rules that contribute to either the minimum or the maximum number of **Instances** at runtime. This means that one and only one instance of the component can exist in the configuration. (See Section 9.14.6, "Instances" on page 9-29.)
- An **instantiable component** is one that may exist once or multiple times in a runtime Oracle Configurator session. One or more instances may exist when the Model is initially loaded, and it may also be possible for the end user to create additional instances by clicking an **Add** button. (For more information, see Section 3.5, "Multiple Instantiation in Solution Models" on page 3-14). Instantiable components may be either:
  - An **optional instantiable component,** which is an instantiable component that has a maximum number of **Instances** of one, and no configuration rules contributing to its maximum number of instances at runtime. This includes a required component (min / max instances = 1 / 1) with a rule that consumes from the minimum number of instances. (In other words, there can be at most one instance of the component in the configuration.)

- A **multiply instantiable component** (that is, one that can be instantiated multiple times) is an instantiable component with a maximum number of **Instances** that is *greater than* one, or an instantiable component that has a maximum of one but a Numeric rule contributes to the maximum at runtime (see Section 5.7.5, "Building Rules Using Node Properties" on page 5-26). In either case, multiple instances of the component can be added to the configuration.

- The **required substructure** of a component consists of the component itself, all of its required children, all of their required children and so on down to, but not including, any instantiable components.

- Components are **independently multiply instantiable** (that is, instantiable multiple times) when the existence of one instantiable component at runtime does not require the existence of the other. In other words, the components are instantiable and exist at the same level in the Model structure (that is, they do not have a parent-child relationship). A configuration rule cannot relate two or more components that are independently multiply instantiable. In Figure 5–24, C1 and C3 are independently multiply instantiable, but C1 and C2 are not.

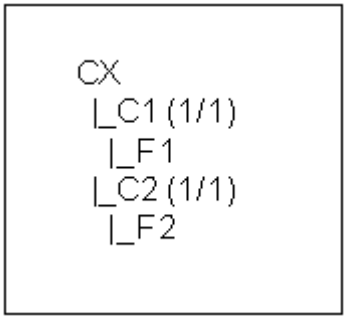
Figure 5–24 Independently Multiply Instantiable Component



Following are some examples of valid rules that relate runtime components:

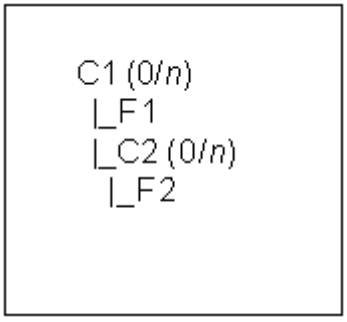
- A rule that relates components within the required substructure of a component. For example, nodes from the structure shown in Figure 5–25 are participants in a Logic rule that states "F1 Requires F2."

Figure 5–25 Components within Required Substructure



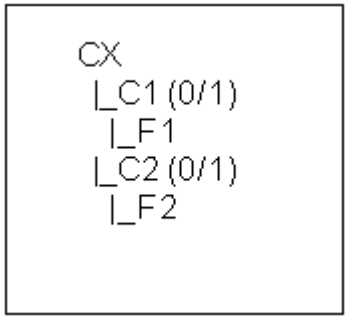
- A rule that relates components within the required substructure of any parent component. For example, C1, C2, and C3 are all components, and C1 contains (is the parent of) both C2 and C3. You can create a rule that relates components C1 and C2 or C1 and C3. However you cannot create a rule that relates C2 and C3 because they are siblings (that is, they exist at the same level in the Model structure). For example, nodes from the structure shown in Figure 5–26 are participants in a Logic rule that states "F1 Requires F2."

Figure 5–26 Components within Parent Component's Substructure



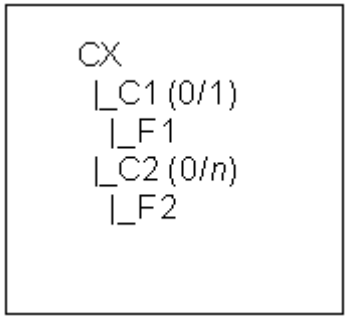
- A rule that relates an optional component with any number of sibling optional components. For example, nodes from the structure shown in Figure 5–27 are participants in a Logic rule that states "F1 Requires F2."

Figure 5–27 Optional Component and Sibling Optional Components



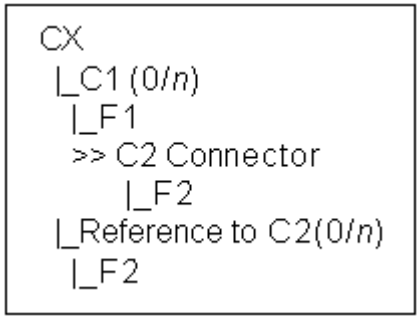
- A rule that relates an optional component with at most one other multiply instantiable component. For example, nodes from the structure shown in Figure 5–28 are participants in a Logic rule that states "F1 Requires F2."

Figure 5–28 Optional Component and Multiply Instantiable Component



- A rule that relates any two components if there is a Connector from one component to the other. In the structure shown in Figure 5–29, the Logic rule "F1 Requires F2" is valid if you use F2 from the structure of the Connector's target Model when defining the rule, but the rule is not valid if you select the node from the referenced Model's structure.

Figure 5–29 Component with a Connector

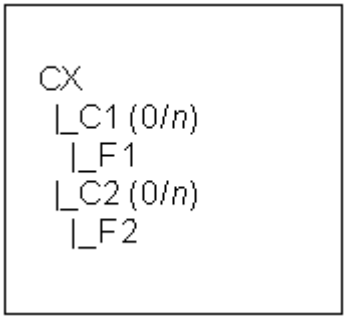


For more information about defining rules using structure of a target Model, see Section 3.6.1, "Connectors and Target Models" on page 3-21.

Following are some examples of unacceptable combinations:

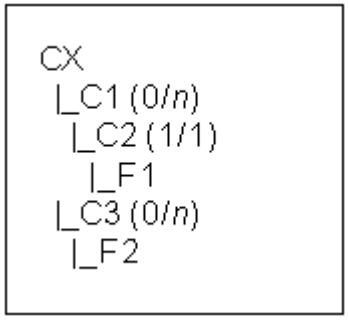
- A rule that relates two multiply instantiable Components at the same level in the Model. For example, the Logic rule "F1 Requires F2" is invalid when the nodes from the structure shown in Figure 5–30 are participants.

Figure 5–30 Sibling Multiply Instantiable Components



- A rule that relates a multiply instantiable component multiple times with a component in the required substructure of a multiply instantiable component at the same level in the Model. For example, the Logic rule "F1 Requires F2" is invalid when the nodes from the structure shown in Figure 5–31 are participants.

Figure 5–31 Sibling Multiply Instantiable Components



6

Constructing the User Interface

When you are satisfied with any part of your Model and want to test it, you need to generate a User Interface with which to access the Active Model. Oracle Configurator Developer can automatically generate a User Interface derived directly from the Model. This generated UI reflects the structure of your Model and provides all the UI elements for implementing a runtime Oracle Configurator. This chapter describes how to generate and customize a User Interface in Configurator Developer.

Following are the basic steps for creating and testing a User Interface:

1. Generate a new UI (see Section 6.2 on page 6-16)
2. Customize the UI, if required (see Section 6.3 on page 6-20)
3. Preview screens using the UI Editor (see Section 6.4 on page 6-41)
4. Test and debug the UI using the Test module (see Section 7.1 on page 7-1)

> **Note:** User Interface design directly affects how a configuration model performs in a runtime Oracle Configurator. For information about how to construct a UI for maximum runtime performance, see the *Oracle Configurator Performance Guide*.

6.1 The User Interface Module

While testing and debugging a configuration Model, you need to update the User Interface to reflect changes in the Model. If your changes are minimal, you can refresh the UI by selecting the *Product Name* User Interface node and then choosing **Edit > Refresh**. If you make extensive changes to the Model, you must create a new

UI. For this reason, it is recommended that you do not customize the User Interface extensively before the Model and your configuration rules are complete.

You must refresh a UI to ensure that any changes you make to the Model structure or Model data appears at runtime. **UI Refresh** is not required if you make changes in the UI module, such as modifying UI Screen attributes, adding Text, Buttons, or Pictures, or changing the Option display settings for individual Features.

Oracle Configurator Developer takes advantage of the Oracle Configurator architecture by storing the complete definition of the User Interface in the CZ schema, where it is available to the Oracle Configurator Developer and the Active Model in a runtime Oracle Configurator.

For information about User Interfaces function within the context of a Model reference, see Section 6.1.8, "Referencing and the Model User Interface" on page 6-15.

6.1.1 Deleting a User Interface

Because developing and customizing a User Interface is an iterative process, you may generate many UIs for the same Model and have some that you no longer need. To delete a UI, go to the UI module, select the UI, and then:

- Choose **Edit > Delete**

  or

- Click the right mouse button, and then choose **Delete**

You *cannot* delete a User Interface if it has been published (that is, it is part of an existing publication record). For information about Publishing, see Section 2.2 on page 2-11.

Additionally, you cannot delete a UI if it is referenced by the Model you are currently editing. (When a Model references another Model, the referenced Model's User Interface(s) are also referenced.) In this case, you must open the Model to which the UI belongs in the Model window, and then follow the steps above to delete it. See Section 2.1.3, "References and UI Definitions" on page 2-3.

6.1.2 How the Oracle Runtime Configurator Displays the Model

Your choice of deployed application influences how a Model appears in the runtime Configurator. The choices are a Java applet or a DHTML window within Oracle Applications or in a custom Web application. Much of how the Model is displayed in the deployed application is common to either deployment method. The following

sections describe some of the differences between the Java applet and the DHTML (Components tree) runtime Configurator.

For a description of the common behavior of any runtime Configurator, see Section 7.1.4, "Configuring an Item in a Runtime Oracle Configurator" on page 7-3.

6.1.2.1 Runtime Oracle Configurator: Java Applet

Only BOM nodes are visible in the Java applet. You cannot modify the Java applet UI. The Java applet ignores how you set the Visibility toggle on Components in Configurator Developer. See Section 6.1.5, "Controlling Visibility" on page 6-10.

In the Java applet, each component is configured in the option selection pane in the upper-right region. The tree of configurable components is in the Model structure pane on the left. End users navigate from one component to another in this Model structure pane. When an end user selects a component, the selection options for that component's features display in the selection pane. Components whose configuration is unsatisfied or incomplete are marked by a red asterisk in the folder icon for that component.

You can launch either a DHTML Configurator window or the Java applet window from Oracle Order Management (OM). The UI type selected by OM (or any other host application) depends on the parameters provided in the host application's *session initialization message*. For more information, see Section 2.2, "Publishing" on page 2-11.

If you add structure to gather customer requirements or to support guided buying or selling questions to your Model, create a Components tree (DHTML) UI for the Model. When the end user is done with the DHTML UI, a small window is displayed with a button to terminate the Configurator. When the end user clicks this button, the Java applet queries the database to get an appropriate termination message, passes the message on to Order Management, and terminates itself.

Note that if you want the end user to make selections from the Java applet UI, your Model must not have required selections that are not part of the BOM. If you have non-BOM items that are required, the configuration cannot be satisfied because they do not appear in the Java applet and therefore cannot be selected by the end user.

The root node of the Model does not appear in the Java applet. Oracle Order Management users can order more than one instance of the Model by specifying a quantity in the Sales Order window.

6.1.2.2 Runtime Oracle Configurator: DHTML

The DHTML window can be fully customized. In any DHTML window, each component is configured on a separate page in the Oracle Configurator window. Selection options may be presented in the form of needs assessment questions the end user answers to provide requirements for the product. End users click the button in the selection pane for advancing to the next component to configure.

Every DHTML window provides buttons that enable the end user to switch from Configuration mode to the Summary screen. The Summary screen displays information about all items selected during the configuration session, such as the item name, description, quantity, and unit of measure. An end user can click the **Summary** button at any time during a configuration session to display this information. When viewing the Summary screen, an end user returns to the configuration session by clicking the **Configuration** button.

You can control whether items display in the Summary page by modifying settings in the CZ_PS_NODES table. If you use a custom program to import data into the CZ schema, review the settings in this table to be sure all desired information appears at runtime. See the *Oracle Configurator Implementation Guide* for more information.

Various options available when creating a new DHML UI enable you to manipulate its overall appearance, generate unique captions for each Model node, create navigation buttons on each page, and display or hide the runtime **Navigation Tree** that is based on your Model structure (see Section 6.1.6.1 on page 6-12). You can also create action buttons in Configurator Developer or write Functional Companions to navigate to specific UI Component pages. See Section 6.3.4.4, "Adding Buttons to a Screen" on page 6-28 and Section 5.14, "Functional Companions" on page 5-67.

6.1.3 How the Model Affects the User Interface

The User Interface you generate is based on your Model. Therefore, the structure of the Model influences the structure of the User Interface in some important ways.

A User Interface is based on a single Model, which can contain sub-Models. Sub-Models can have their own UI or be included in a UI based on their parent Model (for example, if a Model is referenced by another Model). See Section 6.1.8, "Referencing and the Model User Interface" on page 6-15.

Models, sub-Models, BOM Models, BOM Option Classes, and Component nodes are represented by individual screens in the runtime UI, and Configurator Developer creates a separate node for each in the UI tree. Each Feature of a Component or Model node appears as a separate control within its parent UI

screen. The same is true for BOM Standard Items, which appear within the same UI screen as their parent BOM Option Class. You can modify the type of Feature control created for Feature nodes (Dropdown List or Selection List), but BOM Option Classes generate selection lists and the control type is read-only.

The Model tree is "flattened" in the UI tree. Screens all appear at the same level in the UI tree, no matter now deeply nested the corresponding node may be in the Model tree.

The effectivity defined for a Model, Product, or Component also affects the Model's appearance in the runtime UI. For example:

- Model nodes that are not effective due to their effective date or the Usage(s) to which they are assigned do not appear in the runtime UI.
- All Models, Products, Components, and BOM Option Classes have their own pages in the runtime UI. Any pages that are not effective do not appear in the runtime UI.
- Any links, buttons, images, and action buttons that control navigation to an ineffective (hidden) page do not appear in the runtime UI. If you defined a Functional Companion to navigate to a page but that page is not effective in the runtime UI, Configurator Developer displays a message that tells you why it is not available.
- If a Model contains references to another Model and that Model is not effective, none of the referenced objects appear in the UI.

See Section 3.3, "Effectivity" on page 3-4.

6.1.3.1 Data Fields

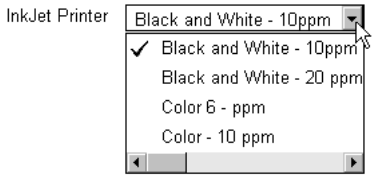
At runtime, Features, Resources, and Totals appear in their parent node's UI screen. For imported BOM information, BOM Option Classes and BOM Standard Items display as fields on the screens of their parent BOM Models and BOM Option Classes. Totals and Resources generate Value Displays.

For Features that you create in Configurator Developer, use the **Data Type** setting to determine the UI Control that appears at runtime. This setting appears in the **Type** attribute in the Model module. For more information about this attribute, see Section 9.14.3, "Type" on page 9-23.

Following are examples of how each Data Type appears in the runtime UI.

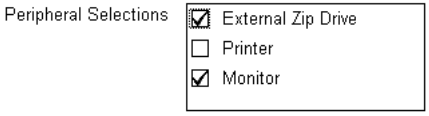
A Feature with a **Data Type** of **List of Options**, a **Minimum** and **Maximum Number of Selections** of 1/1, and no **Counted Options** appears as a dropdown list at runtime (see Figure 6–1).

Figure 6–1 Dropdown List



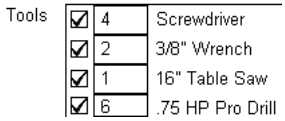
A List of Options Feature with a **Maximum** greater than 1 and no **Counted Options** generates a Selection List with a check box for each option (see Figure 6–2).

Figure 6–2 Selection List



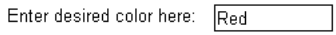
A List of Options Feature with **Counted Options** generates a Selection List with a numeric input field for each Option. This enables the end user to specify a quantity for an item by entering a numeric value. For example, the maximum **Number of Selections** for the "Tools" Feature is set to 4 in Configurator Developer, so the end user can order more than one of each option in the list (see in Figure 6–3). If the **Maximum** were set to 1, the end user could enter a quantity for only one Option in the list.

Figure 6–3 Selection List with Input Field



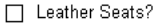
Features with a **Data Type** of **Integer Number**, **Decimal Number**, or **Text** generate an input field (see Figure 6–4). The type determines the value the end user can enter. For example, a **Data Type** of **Decimal Number** allows end users to enter a value such as 12.5.

Figure 6–4 Text Input Field



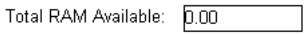
Features with a **Data Type** of **True/False** (Boolean) generate a check box (see Figure 6–5). If the **Default Value** is set to **True**, the box is selected when the configuration session begins; otherwise, the box is empty (not selected).

Figure 6–5 True/False Feature Data Type



Totals and Resources display as a read-only text box (see Figure 6–6). The value of a Total or Resource is generated by one or more Numeric rules that you create.

Figure 6–6 Read-Only Text Box



6.1.3.2 Data Field Labels

Configurator Developer automatically generates labels for data fields for all Features, Resources, and Totals. You can choose to create these labels using the node name, description, or by concatenating both the name and description. See Section 6.2, "Generating a New User Interface" on page 6-16.

The display area for labels is a fixed, predetermined size. This default size may not be the best for your runtime Oracle Configurator, so be sure to check the appearance of labels in your generated UI screens. If necessary, you can alter the size of the label display area. See Section 6.3, "Customizing the Default User Interface" on page 6-20.

6.1.3.3 Add and Delete Buttons

If the Minimum and Maximum number of **Instances** for a Model or Component are not equal, Configurator Developer provides an **Add** button on the Model or Component's parent UI screen (see Figure A.2 on page A-6). An Oracle Configurator user can click this button to add instances of a Model or Component to the configuration.

An end user cannot add more **Instances** of a component than the maximum specified in Configurator Developer, unless a Numeric rule increases the maximum allowed based on end user selections. See Section 5.7.5, "Building Rules Using Node Properties" on page 5-26.

When multiple instances of a Model or Component exist within a configuration, the runtime UI dynamically generates a **Delete** button on the Model or Component's UI screen. The end user clicks this button to remove the selected instance from the configuration.

By default, these buttons are labeled **Add** and **Delete** when you generate a new UI, and their associated actions are Add Component and Delete Component, respectively. However, you can modify the label and action of each button if necessary. For more information, see Section 6.3.4.4 on page 6-28.

Note: The runtime UI does not generate the **Delete** button when the selected Model or Component represents the top-level UI for a configuration session.

Because a Model can **reference** another Model, Configurator Developer also displays a **Delete** button on each UI screen that represents a referenced Model. If a Model is referenced by another Model, and the referenced Model is *not* required in the configuration (its minimum number of **Instances** is 0), the UI server displays a **Delete** button in the Model's UI screen. This button does *not* appear if the referenced Model is required and only one instance of it exists in the configuration, or if it is the top level Model being configured. If the minimum and maximum number of **Instances** of a referenced Model are not equal, the Oracle Configurator Servlet generates an **Add** button and displays it in the parent Model's UI screen. (For more information about the **Instances** attribute, see Section 9.14.6 on page 9-29.)

Example: Model M1 references M2 and the Reference has a min/max of 0/5. When you generate the UI for M1, the UI screen for M1 includes an **Add** button that an end user can click to add instances of M2 to the configuration. When an instance of M2 is added to the configuration, the runtime configurator displays a **Delete** button

on the UI screen for the new instance of M2. The end user can optionally click this button to remove the selected instance of M2 from the configuration.

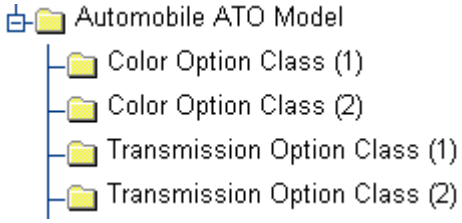
For more information about Model referencing, Section 2.1 on page 2-1.

6.1.3.4 Feature Options and BOM Standard Items

After generating a new UI, you can sort and display Feature Options based on the Option name, a common Property value, or each Option's location in the Model structure. Like many other UI options, you can choose a default Option sorting method at the UI level (such as alphabetically in descending order), and then enter a different method for specific Features. For more information, see Section 6.3.1, "Customizing the User Interface Default Settings" on page 6-21.

BOM Option Classes containing many Items can generate large UI screens and may be difficult for end users to navigate at runtime. Therefore, you may want to create smaller groups of Items and display them on separate screens. Example: The Color Option Class has 20 Items and you specify a maximum number of 10 Items per screen. Configurator Developer creates two UI screens for this Option Class and displays 10 Items on each. If you choose to display the Navigation Tree, the runtime UI displays a separate link for each screen and numbers them sequentially, as shown in Figure 6–7.

Figure 6–7 Runtime Navigation Tree for Multi-Screen BOM Option Classes



You can specify the maximum number of Items to display on each UI screen when creating a new DHTML UI. See Section 6.2, "Generating a New User Interface" on page 6-16.

6.1.4 Pricing and ATP Display

You can optionally display Available to Promise (ATP) dates, the list price, selling price, and extended price for Items in both the Java applet and the DHTML

window. Pricing information is available only for Items that are defined in the Item Master. See Section 3.1, "The Item Master" on page 3-1.

If pricing is enabled, this information appears in the runtime UI automatically, but the end user must click a button that you create in Configurator Developer to display ATP dates. The Java applet displays the list price, selling price, and extended price by default. The total price is updated when the end user clicks the **Update** button. The system updates the extended price (quantity * unit price) when the end user clicks the **Update** button or makes another selection.

In the DHTML window, pricing information appears on each Component page and the Configuration Summary page. You can control whether the DHTML window displays the list price or the selling price for each Item using the Pricing Display attribute in Configurator Developer. See Section 6.1.4.1, "Customize Pricing Display in the DHTML Window" on page 6-10.

Note: If an item's description is very long, pricing information may overlap the description in a DHTML UI Summary screen. Consider this possibility when designing your Model's User Interface.

To enable the display of pricing and ATP information in the runtime UI:

- Define system properties for the OC Servlet. See the *Oracle Configurator Installation Guide* for more information.
- Define parameters for the host application's initialization message. See the *Oracle Configurator Implementation Guide* for more information.

6.1.4.1 Customize Pricing Display in the DHTML Window

Use the **Pricing Display** attribute to customize how prices appear and are updated in the DHTML window. This attribute consists of two settings: **Item Price Display** and **Price Update**. You specify these settings when modifying the UI settings for your configuration Model. See Section 6.3.1, "Customizing the User Interface Default Settings" on page 6-22.

6.1.5 Controlling Visibility

There may be portions of a Model's structure that you do not want to display in the User Interface. For example, you define a Component that is a collection of Totals used in calculations required by the configuration. You use these nodes when

testing your runtime Configurator, but do not want your end users to see them. The **Visibility** attribute determines whether the Component or any of its child nodes appear in the runtime UI.

To prevent a node from appearing in the runtime User Interface, deselect the **Display in User Interface** box. This option appears in the **Visibility** attribute in the Model window. You can set the **Visibility** attribute at the UI level for any Model node. You can also dynamically hide Features, BOM Option Classes, and their children based on their availability in the runtime UI. See Section 6.1.5.1, "Dynamic Visibility" on page 6-11.

6.1.5.1 Dynamic Visibility

Use *dynamic visibility* to determine whether UI controls and individual options appear in the runtime DHTML UI when their logic state changes to Logic False in the current configuration session. For example, you create a rule in Configurator Developer that states "Option1 Excludes Option3." If the **Hide unselectable Controls and Options** option is enabled, Option3 no longer appears in the UI when the end user selects Option1. However, if the end user deselects Option1, Option3 is once again visible in the UI.

You specify a default Visibility setting for a generated User Interface and each screen inherits this setting. See Section 6.3.1, "Customizing the User Interface Default Settings" on page 6-21. However, you can optionally override the default setting for individual Features, BOM Option Classes, and their children. See Section 9.14.5, "Visibility" on page 9-53.

> **Note:** **Hide unselectable Controls and Options** and **Hide when unselectable** can produce unintended results at runtime. Example: An Option is set to Logic False by a Defaults rule and **Hide when unselectable** is enabled. As a result, the end user cannot override the rule and select the Option because the Option no longer appears in the UI. However, you can use the OC Servlet parameter `lfalse_is_not_available` to modify the runtime behavior of Logic False Options when either of these options are enabled. For more information, see the section on Oracle Configurator Servlet properties in the *Oracle Configurator Installation Guide*.

Things to Consider

Note that there may be times when an option has a logic state of Logic False but it is still available for selection. This can occur because of a Defaults configuration rule

or a Maximum Selections relation where the maximum is greater than one. In these cases, the option may have a logic state of Logic False but can still be available in the UI. As a result, it appears in the UI even if **Hide when unselectable** or **Hide unselectable Options** is enabled.

Also note that because Items occupy a fixed position and size on a page, hiding them using Effectivity or Dynamic Visibility can create "gaps" in the UI (this does not occur when hiding Feature Options, however). Consider this when choosing to hide Items using these methods and design each page accordingly. For example, you might group objects that may be hidden in the runtime UI towards the bottom or side of a page.

If you change the default name of a Reference node in Configurator Developer, the name of the item appears differently in the Summary screen and in the host application. See Section 2.1.8.2, "Renaming Reference Nodes" on page 2-8.

6.1.6 Runtime Navigation

The ability to view all available options, make selections, and configure a valid, orderable product depends upon the end user's ability to successfully navigate the runtime UI. Configurator Developer provides the following types of navigation options, depending on the UI style you select, to ensure that all options within your configuration model are accessible:

- The runtime Navigation Tree
- Navigation buttons

6.1.6.1 Runtime Navigation Tree

The runtime Navigation Tree is available in both the Java applet and the DHTML UI and is based on the structure of your Model. BOM Model, BOM Option Class, and Component nodes appear as folders in the Navigation Tree and represent UI screens from which the end user selects options. If you want, you can also choose not to display the Navigation Tree in the runtime UI. (See Section 6.2.1, "Display Width Reserved for Navigation Frame" on page 6-20.) However, if the UI does not include a Navigation Tree you must create navigation buttons so that end users can access all available configuration options.

An example of the runtime Navigation Tree is shown in Figure 6–7 on page 6-9.

For information about customizing the runtime Navigation Tree, see Section 6.3.2 on page 6-24.

6.1.6.2 Navigation Buttons

Runtime *navigation buttons* also provide end users with easy access to each screen in the DHTML UI.(Navigation buttons are not available in the Java applet.) Configurator Developer provides two types of navigation buttons:

- Wizard-style buttons
- Action buttons

If you enable the appropriate option when generating a new UI, Configurator Developer automatically generates Wizard-style navigation buttons and displays them on every screen. These navigation buttons consist of a Next, Back, and Home button which appear at the bottom of each screen and enable end users to go to the previous, next, and top-level screens in the UI. By default, the Back and Home buttons do not appear in the first UI screen, and the Next button does not appear in the last screen. Additionally, this type of button may be dynamically hidden at runtime based on end user selections. Example: The user is viewing the next to last screen in the UI and makes a selection that causes the last UI screen to be hidden. When this occurs, the Next button disappears since the current screen is now the last screen in the UI.

You can modify the text and action of Wizard-style navigation buttons as necessary, and cut, copy, and paste them as you would any other UI object (this preserves the ability to dynamically hide these buttons at runtime, as described above). You can also choose to display these buttons instead of or in addition to the runtime Navigation Tree. See Section 6.2, "Generating a New User Interface" on page 6-16.

Action buttons are similar in many ways to Wizard-style navigation buttons, but you create this type of button manually on specific screens after generating a UI and indicate which screen to display when the end user clicks on it at runtime. Navigation actions for buttons include go to the next screen, go to the previous screen, or go to any screen in the UI that you specify. You may want to create action buttons in addition to Wizard-style buttons to further enhance usability of the runtime UI. See Section 6.3.4.4, "Adding Buttons to a Screen" on page 6-28.

Use the UI Editor to view and reposition either type of navigation button on a screen. Use the UI module to modify a button's action and label text.

6.1.7 Multiple Language Support and the Model User Interface

If you have implemented Multiple Language Support (MLS), you can create a single UI to use with multiple languages or create a different UI for each language in which you do business. For more information, see Section 1.5, "Multiple Language Support in Oracle Configurator" on page 1-10.

Some changes may be required if you want to use the same UI with multiple languages. See Section 6.3.4.1, "Customizing a UI that Supports Multiple Languages" on page 6-26.

After unit testing the Model, configuration rules, and UI using the Test module, publish the Model and UI to make them available to a hosting application in one or more languages. See Section 2.2.2, "Publishing and Multiple Language Support" on page 2-24.

6.1.7.1 User Interface Captions

Configurator Developer always uses the *UI caption* to label a BOM Standard Item or Feature Option at runtime. Depending on the setting of the **Generate Captions From** option, Configurator Developer creates default UI captions either from the node name, node description, or both when you generate a new UI. Therefore, the text used to display an option at runtime may be different than the description entered in Oracle Applications (for BOM Standard Items) or in Configurator Developer (for Feature Options). Configurator Developer provides several methods you can use to modify the default UI captions of individual Features, Feature Options, and BOM Standard Item. For more information, see Section 9.21.8, "Option Display" on page 9-50.

6.1.7.2 BOM Item Descriptions

Oracle Inventory users can enter alternate translations for each item description using that application, but the item *name* exists only in the base language of the Oracle Applications installation. However, both the name and description are imported into Configurator Developer with the BOM. If you need to create one UI that will be used with multiple languages, do not choose to use the node name when creating a new UI. Because the same UI may be displayed in one of several different languages (based on the language specified by the hosting application at runtime), captions that include the node name may confuse end users (for example, an English item name with a French description). Therefore, when using MLS, it is recommended that you generate UI captions using only node *descriptions*. See Section 6.2, "Generating a New User Interface" on page 6-16.

6.1.7.3 Violation Messages

Configurator Developer checks the setting of the **Generate Captions From** option when creating the violation message that appears at runtime. For example, if you choose to generate UI captions using both the node name and description, the violation message at runtime displays both the node name and description of the Feature, Option Class, or Option that caused the violation.

For more information, see Section 1.5, "Multiple Language Support in Oracle Configurator" on page 1-10.

6.1.8 Referencing and the Model User Interface

At runtime, the UI of a given Model can be invoked either directly or as part of the UI for another Model that references it. If the UI is invoked by reference, and the corresponding Model reference can be **instantiated**, the Generate UI procedure creates a button to delete the current instance of the referenced Model in the root screen of the referenced UI. However, Configurator Developer does not display this button at runtime if the UI is invoked as the top-level UI for a configuration session. See Section 6.1.3.3, "Add and Delete Buttons" on page 6-8.

If a referenced child Model has no compatible UI defined, Configurator Developer generates one in the child when you generate a new UI for the parent.

6.1.8.1 Model UI Scope and UI References

Each UI contains only the node and page definitions for the Model to which it belongs. When a Model references another Model, the UI of the parent Model contains a node that identifies one of the referenced Model's UIs. If a referenced Model has multiple compatible UIs available, Configurator Developer chooses the latest compatible UI when generating the UI for the parent Model. At runtime, Configurator Developer creates the complete UI for the parent Model.

The parent Model's default UI settings always take precedence over the default UI settings defined for any referenced Models. However, depending on their setting, individual Features or BOM Option Classes within a referenced Model may either inherit or override the *parent Model's* default UI values. For example, Model A references Model B and Model C. When you generate the UI for Model A, all of the screens within Models A, B, and C derive their default settings from the UI for Model A (the parent Model). All Features and BOM Option Classes within Model B and Model C that have **Visibility** or **Option Sorting** methods set to **Use default** also inherit Model A's default UI settings. To modify UI settings for Features and BOM Option Classes that belong to a referenced Model, you must open the referenced Model in the Model window, go to the UI module, and change the setting for that UI node.

Because a Model may have multiple UIs, you can specify which UI to use for a referenced child Model after generating the parent Model's UI.

To select a different UI for a referenced Model:

1. Go to the UI module.

2. Expand the parent Model's UI node, then select the referenced Model's UI node. (For example, the node labeled "Referenced Model A User Interface 2".)
3. In the Attributes view, click the **Change** button.
4. Select a different UI from the Target UI dialog.
5. Click OK.

6.1.8.2 UI Generation

If a referenced child Model has no UI defined, Configurator Developer generates a UI for the child Model, then proceeds with generating a UI for the parent Model. Configurator Developer stores the child Model's UI definition in the child Model, not in the parent.

6.1.8.3 Updating a Referenced UI

To make changes to a referenced UI, you must first open the Model to which it belongs in the Model window. You do not need to refresh the parent Model's UI after making changes to a referenced UI. Because the UI for a referenced Model is *linked* to rather than incorporated into the parent Model's UI, any changes to the referenced UI are reflected in the parent Model's UI automatically.

Configurator Developer prevents you from *deleting* a UI that is referenced by another Model, since doing so would cause screens to be missing from the parent Model's UI.

6.1.8.4 Publishing

A broken or missing UI Reference link as described above results in a "consistency check" error when publishing a Model or updating an existing publication. A broken or missing UI Reference link prevents either operation from proceeding.

6.2 Generating a New User Interface

When creating a new User Interface for a configuration model in Configurator Developer, you can choose to generate either a BOM Model Tree or Components Tree UI. Choose BOM Model Tree to create a Java applet UI. Choose Components Tree to create a customizable DHTML UI. For more information about these UI styles, see Section 6.1.2, "How the Oracle Runtime Configurator Displays the Model" on page 6-2.

The New User Interface dialog enables you to specify a UI style and, if you choose **Components Tree**, provides additional parameters you can use to customize its

appearance. When you create a BOM Model Tree UI, these additional parameters are not available.

After generating a new UI, use the UI Editor to preview the layout of each screen and reposition UI controls, buttons, and so on so they will appear as desired at runtime. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

To generate a new User Interface:

1. Select the **UI** button on the main toolbar to go to the User Interface module.
2. Choose **New User Interface** from the **Create** menu. Alternatively, you can highlight the User Interfaces node in the UI Context Tree View, select the right mouse button, and select **New User Interface** from the pop-up menu.
3. A New User Interface dialog appears, prompting you for additional information to be used in generating the User Interface. This dialog contains the following controls:

   - A list of the available **UI Style** options. The default value is **Components Tree**. Select this option if you are deploying your configuration model as DHTML in a browser. Select **BOM Model Tree** to display your deployed configuration in a Java applet. Remember that if you select **BOM Model Tree**, you cannot customize the UI.

     Warning: Do not create a BOM Model Tree UI for a non-BOM Model. If you do this, the following error message appears when you attempt to view the Model at runtime: "WebUI Initialization Failure: User interface is not compatible with the client type."

   - A list of the available **Look and Feel** settings. Use this list to specify the color scheme, controls, and layout for your Components Tree (DHTML) UI. Select **Oracle Web Look** to generate a UI similar to Oracle's Web based applications, such as Oracle *i*Store. Select Oracle Forms Look to generate a UI similar to the Oracle Applications user interface, such as Order Management. The setting you choose determines the look and feel for all nodes in the UI, including any References.
   - A list of choices for the **Layout Area**. Configurator Developer creates a window based on this value and adjusts the layout of the UI to fit within the space available. The choices are:

     * 640 x 480 (default)

- 800 x 600
- 1024 x 768
- 1600 x 1200

- Use the **Generate Captions From** list to display Options and BOM Standard Items at runtime using the node name, description, or by concatenating both the name and description. The default is **Description**. You can override this setting for individual BOM Standard Items and Options after generating the UI. See Section 9.21.8, "Option Display" on page 9-50.

  By default, Configurator Developer separates each text string using a comma and two spaces when you generate UI captions from the node name and description. (For example: "AT62431, Sentinel Custom Laptop".) You can modify the character Configurator Developer uses to separate each string by running the Modify Configurator Parameters concurrent program. For more information, see the *Oracle Configurator Implementation Guide*.

  If you have Multiple Language Support (MLS) installed and need to publish Models in multiple languages, set **Generate Captions From** to **Description**. See Section 6.1.7, "Multiple Language Support and the Model User Interface" on page 6-13.

- Select the **Wizard-Style Navigation** check box to display **Back**, **Next**, and **Home** navigation buttons on each UI screen. For more information, see Section 6.1.6, "Runtime Navigation" on page 6-12.

- The **Show all nodes box** overrides all **Display in User Interface** settings in the Model. If this box is checked, all nodes in the Model structure are displayed in the UI, regardless of how you set the Visibility option for each node. By default, this box is unchecked, which means Configurator Developer uses the Visibility setting defined for each node to determine whether to create a corresponding screen or UI control for that node.

- Use the **input box** to specify the **Percentage of layout area reserved for the navigation frame** in the runtime UI. Possible values are 0 to 50. Enter 0 if you do not want to display the Navigation Tree. See Section 6.2.1, "Display Width Reserved for Navigation Frame" on page 6-20.

- Use the **Maximum Number of BOM Option Class Children per Screen** input box to control how many Items within a BOM Option Class appear on each UI screen. You may want to specify a maximum number of Items to appear on each UI screen to improve usability if, for example, some Option Classes contain many Items. See Section 6.1.3.4, "Feature Options and BOM Standard Items" on page 6-9.

4. When you click **OK**, an algorithm creates User Interface nodes corresponding to the nodes in the Model. The User Interface structure is displayed in the User Interface Tree.

   The top-level node of the new User Interface structure has a default name generated using the Model node name followed by "User Interface" (for example, PT3241 User Interface). You can edit the name and description of the UI in the usual fashion. To unit test the runtime User Interface, click the **Test** button. For more information, see Section 7.1.3, "Testing your User Interface" on page 7-3.

Table 6–1 lists the available elements of the Generic User Interface.

Table 6–1 Elements of the Generic User Interface

Element	Description
User Interface	This is the root node of the UI.
Components Tree	This node contains default values for displaying the Navigation Tree in the runtime UI. See Section 6.3.2, "Customizing the Display of the Runtime Navigation Tree" on page 6-24.
Screen	This node groups the nodes that appear together on a single screen in the UI and corresponds to a BOM Model, BOM Option Class, or a Component node. You can add Buttons, Pictures, and Text to UI screens. See Section 6.3.4, "Customizing Screen Design" on page 6-25.
Picture	This node represents a graphic image file. Add graphics to improve the appearance of the UI. You can also associate an action with an image. See Section 6.3.4.2, "Adding Graphics to a Screen" on page 6-26.
Text	This node represents text that you add to the UI. (Note that this is not the UI caption generated for labels and UI controls.) See Section 6.3.4.3, "Adding Text to a Screen" on page 6-27.
Button	This node represents a button in the UI. The end user selects a button in order to perform some action. Section 6.3.4.4, "Adding Buttons to a Screen" on page 6-28.
Feature Control	This node represents a Feature in the UI. The end user uses Feature controls to input values for Features or select from Options of a Feature. See also Section 6.3.4.6, "Sorting Feature Options" on page 6-33.

Table 6–1 (Cont.) Elements of the Generic User Interface

Element	Description
Value Display	This node represents Totals and Resources in the UI. Value Display fields display values to the end-user, and are read-only.
Tagged Value Display	This node specifies data to be displayed in the UI. Choices are Total Selling Price or Total List Price.
Connector	This node represents a UI control that enables the Oracle Configurator end user to connect two instantiated components at runtime. Section 4.3.5, "Creating a Connector" on page 4-6.
Connections Listbox	Section 6.3.4.5, "Adding a Connections Listbox to a Screen"This node represents a box that appears in the parent node's UI screen and displays a list of all components that are connected to the selected component. See also on page 6-30.
Referenced UI	This node specifies the UI for a referenced Model node. See Section 2.1.3, "References and UI Definitions" on page 2-3.

6.2.1 Display Width Reserved for Navigation Frame

When creating a Components Tree (DHTML) UI, you can specify the size of the tree control relative to the display width of the browser window. In the New User Interface dialog you can specify how much screen area is available for the navigation frame in the deployed runtime UI. (This is the frame that contains the Navigation Tree at runtime.) You can enter an integer value between 0 and 50 inclusive. The default setting is 30. Set this option to 0 (zero) if you do not want the Navigation Tree to appear in the runtime UI.

You can override the display width at any time by modifying this setting in the Defaults attribute for the selected UI. Overriding the value here does not affect any existing or new control layouts (for example, by refreshing the UI), but doing so will set the allocation for the tree frame in the runtime UI.

See Section 6.2, "Generating a New User Interface" on page 6-16.

6.3 Customizing the Default User Interface

The default User Interface saves you a lot of time and routine work, but it may not satisfy all the requirements that you defined for your runtime Oracle Configurator User Interface. If you are deploying your configuration model as Dynamic HTML in a browser, you can modify the generic User Interface to meet the unique needs of

your Configurator. Remember that you cannot customize User Interfaces generated for the Java applet. See Section 6.2, "Generating a New User Interface" on page 6-16.

Any customizations that you make to the default User Interface are stored in the CZ schema, so they are available to your running Oracle runtime Configurator.

Some of the many customizations you can make to your User Interface include:

- Customizing the User Interface Default Settings on page 6-21
- Customizing the Display of the Runtime Navigation Tree on page 6-24
- Customizing Screen Display Settings on page 6-25
- Customizing Screen Design on page 6-25

6.3.1 Customizing the User Interface Default Settings

Since you can define multiple User Interfaces for each Model, you may want different default settings for each UI. Each User Interface has default settings that affect all screens in the UI. However, you can override some of these settings for specific UI screens as necessary. You may want to customize these settings specifically for your enterprise to provide a unified appearance to each User Interface.

For example, you can choose to display options in the order that they appear in your Model structure and display a custom picture on the background of every screen in your runtime Configurator UI. You can then choose a different option sorting method for specific Features of the Model as required.

The default UI settings determine how to display:

- BOM UI borders
- font
- screen background color and picture
- logic state
- Feature Options (display order)
- the runtime Navigation Tree

After modifying a UI, you can view the changes using the UI Editor. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

6.3.1.1 Modifying Default UI Settings

To modify the default User Interface settings:

1. Select the **UI** button on the main toolbar.
2. Select the node for the User Interface you want to modify.
3. In the Attributes View in the right pane, open the **Defaults** section if it is closed.

- **Look and Feel** is a read-only field that displays the setting you specified when generating this UI. See Section 6.2, "Generating a New User Interface" on page 6-16.

  If the generated UI style is Components Tree, use the **BOM UI Borders** list to choose the type of border to display around BOM item controls at runtime. Choose **Fixed Single**, **3D**, or **None**. (If the generated UI style is BOM Model Tree, this field is set to None and is read-only. If you chose the Oracle Forms Look setting when generating the UI, the default border style is 3D.)

  You can view the runtime look and feel of all BOM UI borders using the UI Editor. See Section 6.4 on page 6-41.

  Use the **Font** field to select a default Font for each screen. See Section 9.21.6.3, "Font" on page 9-45.

- In the **Screen Background** region, use the **Color** and **Picture** fields to specify the background color and image file for each UI screen. See Section 9.21.6.6, "Background Color" on page 9-46 and Section 9.21.6.4, "Picture" on page 9-45.

- In the **Logic State Display** region, select **Icons**, **Text color**, or both to determine how the User Interface indicates the logic state of options at runtime. By default, the runtime UI displays the logic state of options using icons. If you do not want to indicate logic state using icons or specific colors, deselect both options. You can display logic state using icons, by varying the color of label text, or a combination of both. However, displaying logic state using text color is supported only in the Java applet window. For more information on logic state, see Section 5.1.2, "Configuration Rules and Logic State" on page 5-2.

  To modify the icon used to indicate logic state at runtime, click the **Select** button. The Logic State Icons dialog displays the default graphics used in the runtime Oracle Configurator to indicate each logic state and the name of the associated graphics file for each. You can modify the image to use for check boxes, radio buttons, or icon-style logic display. To do this, select the appropriate tab in the Logic State Icons dialog, then click the dialog button (**...**) to open the standard Windows Open dialog. Select the new image file to use, click **Open**, and then click **OK**. The default graphics files are stored in the

%ORACLE_HOME%\Osp\Shared\ActiveMedia\ directory on the machine running Configurator Developer.

Note: You can optionally use your own graphics files and store them wherever you want when designing and unit testing a UI. These graphics appear in the UI Editor to help you customize the appearance of each UI screen. To appear in the runtime UI, however, image files must also exist in the OC Servlet directory specified by OA_MEDIA. For more information, see the *Oracle Configurator Implementation Guide*.

Select the **Text Color** box to indicate logic state using the color of label text. Click **Select...** to open the Logic Text Colors dialog. The dialog displays the currently selected color for each logic state. For each logic state field, use the **Color** field to select a color. See Section 9.21.6.6, "Background Color" on page 9-46.

Select **Hide unselectable Controls and Options** to hide options and controls that have a logic state of Logic False in the runtime UI. See Section 6.1.5, "Controlling Visibility" on page 6-10.

- In the **Option Sorting** region, choose a **Sort Method** and specify how you want to list selectable Options:
  - Choose **By Order In Model** to display Feature Options in the same order that they appear in the Model Tree view in Configurator Developer. If you change the order in which Options appear in the Model in Configurator Developer, you must refresh the UI to make the changes visible in the runtime UI.
  - Choose **By Label** to list Options alphabetically by their UI caption, then specify whether to sort them alphabetically in either Ascending or Descending order. (UI captions that consist of numbers will be sorted numerically.)
  - Choose **By Property Value** to sort Options alphabetically or numerically by Property value, then specify the Property to use for each Option and a sort order of either **Ascending** or **Descending**. This list displays all Properties defined in the CZ schema. The order is determined alphabetically, numerically, or alphanumerically, depending on the Property value. Options that do not have the Property you specify appear at the end of the

list by their order in the Model when you choose a sort order of Ascending; otherwise, they appear first in the list.

- Select the **List Selectable Options First** check box to display available Options first, followed by Options excluded from the configuration due to a previous selection. The runtime UI reorders lists as necessary when the end user makes a selection.

You can override any of the Option Sorting settings for specific Features. See Section 6.3.4.6, "Sorting Feature Options" on page 6-33.

- In the **Runtime Message Style** region, choose whether to display a node's full path in messages that appear at runtime. For example, if Option 1 appears in a contradiction message and this setting is enabled, the message includes the Option's location in the Model structure (such as Component A: Feature X : Option 1). To display a node's full path in runtime messages, select the **Display node names in Runtime messages with Full Path** box.
- Use **Allocate % of Display Width to Navigation Frame** to specify the percentage of the total screen width allocated to the runtime Navigation Tree. Enter a value from 0 to 50, inclusive. Enter 0 if you do not want to display the runtime Navigation Tree. See Section 6.2.1, "Display Width Reserved for Navigation Frame" on page 6-20.

4. Use the options in the **Pricing Display** attribute to specify which prices appear in the runtime UI and how often they are updated. For more information, see Section 9.21.4, "Pricing Display" on page 9-44.

6.3.2 Customizing the Display of the Runtime Navigation Tree

When you generate a new Components Tree (DHTML) UI, the "Components Tree" UI node contains default settings for the appearance of the runtime Navigation Tree. These settings include the background color and picture, font, and the type of border to display. You may want to modify some of these settings to customize the appearance of the Navigation Tree.

For more information about the runtime Navigation Tree, see Section 6.1.6.1 on page 6-12.

The runtime Navigation Tree is *not* customizable in the Java applet UI.

To modify the appearance of the runtime Navigation Tree:

1. Go to the UI module.
2. Expand the User Interface Tree and select the UI to modify.

3. Select the **Components Tree** node.
4. In the Attributes View, expand the **Definition** attribute.
5. Modify any of the following settings as required:
   - Select a font in the **Font** field. See Section 9.21.6.3, "Font" on page 9-45.
   - Specify a background image with the **Background Picture** field. See Section 9.21.6.4, "Picture" on page 9-45.
   - Use the **Background Color** field to specify a color. See Section 9.21.6.6, "Background Color" on page 9-46.
   - Specify a border for the Navigation Tree using the **Borders** field. Select None or Fixed Single.
6. To view your changes at runtime, refresh the UI, then click the **Test** button.

6.3.3 Customizing Screen Display Settings

When you create a new User Interface, the User Interface tree is populated with screen nodes that directly correspond to each Product, Component, BOM Model, and BOM Option Class node in your Model. Each screen node uses the background color, background picture, and font specified at the UI level, but you can choose different settings for individual UI screens to customize their appearance.

6.3.3.1 Changing the Screen Display Settings

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface Tree, then select the UI node of the screen you want to modify.
3. In the Attributes View in the right pane, open the **Styles** section if it is closed.
   - Use the **Background Color** field to specify a color. See Section 9.21.6.6, "Background Color" on page 9-46.
   - Use the **Background Picture** field to specify a background image. See Section 9.21.6.4, "Picture" on page 9-45.
   - Use the **Font** field select a Font. See Section 9.21.6.3, "Font" on page 9-45.

6.3.4 Customizing Screen Design

Each screen in your User Interface contains nodes for the title text. You can customize screens by modifying the title text, by adding pictures, text, and buttons,

and by modifying the display of Features, BOM Standard Items, Totals, and Resources.

6.3.4.1 Customizing a UI that Supports Multiple Languages

If you have implemented MLS and want to use the same UI with multiple languages, be sure to review and modify the layout of the screen as necessary using the UI Editor. Some changes may be needed because the organization and content of the UI may vary depending on the desired language. For example:

- The length of text descriptions can vary significantly between languages
- Some languages are read from right to left, rather than left to right (for example, Arabic)

See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

If you have implemented MLS, be sure that all alternate translations of Item descriptions are entered in Oracle Inventory before importing BOM Models into the CZ schema. This reduces the amount of work required when extending the Model structure and creating rules in Configurator Developer.

For more information about MLS, see Section 1.5, "Multiple Language Support in Oracle Configurator" on page 1-10.

6.3.4.2 Adding Graphics to a Screen

To add or customize screen graphics:

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface node to modify.
3. Select the screen node you want to customize.
4. Click the right mouse button, then select **New Picture** from the menu. (Alternate method: Select **New Picture** from the **Create** menu.)
5. Enter a name for the graphic the Attributes View or by renaming the node in the UI Tree view.
6. If you want, expand the **Description** attribute and enter a description of the image.
7. Expand the **Definition** attribute, then:
   - Click the dialog button (**...**) in the **Picture** field to open the standard Windows Open dialog. Select a GIF or JPEG image located in %ORACLE_

HOME%\Osp\Shared\ActiveMedia\ on the machine running Configurator Developer.

Note: Oracle Configurator Developer supports only GIF (*.gif) and JPEG (*.jpg) files in the runtime DHTML UI. Bitmap files (*.bmp) are not supported.

To appear in a runtime Configurator, the image file must also exist in the OC Servlet directory specified by OA_MEDIA. See the *Oracle Configurator Implementation Guide* for more information.

- In the **ALT Text** field, enter text that you want to appear when an end user places the cursor over the graphic in the runtime Configurator. For example, you may want to provide a description of the graphic or the action the end user can perform by clicking on it (if any).
- Select **Fixed Single** from the **Borders** list if you want a display a border around the picture. See Section 9.21.6.5, "Borders" on page 9-46.
- If you want the Oracle runtime Configurator to perform an action when an end user clicks on the picture, select from the **Action** list. For a description of each action, see Section 9.21.6.7, "Action" on page 9-46.

8. Use the **Layout** attribute to specify the location of the graphic on the screen and specify a **Tab Order**. See Section 9.21.10, "Layout" on page 9-52.

Note: The recommended method of modifying the layout of a UI screen is to use the UI Editor. This tool enables you to drag and drop UI controls using the mouse. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

6.3.4.3 Adding Text to a Screen

To add or customize text on a UI screen:

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface Tree and select the root node of the UI to modify.
3. Select the UI screen you want to customize.
4. Click the right mouse button and then select **New Text** from the menu. (Alternate method: Select **Create > New Text**.)

5. Enter a **Name** for the new text node in either the Attributes View or the UI tree view.
6. If you want, expand the **Description** attribute and enter a description of the text.
7. Expand the **Label** attribute, then enter the text that you want to appear in the UI in the **Text** field. This is the node's *UI caption*. For more information, see Section 6.1.7.1, "User Interface Captions" on page 6-14
8. Use the following settings to modify the runtime appearance and behavior of the text:
   - **Font** (See Section 9.21.6.3, "Font" on page 9-45)
   - **Background Color**: This setting applies only if you set **Background Style** to Opaque. (See Section 9.21.6.6, "Background Color" on page 9-46.)
   - **Background Style**
   - Align
   - **Action**: Use this setting to assign an action to the selected object's UI caption. (See Section 6.3.6, "Assigning an Action to Text" on page 6-39.)

   For more information about these settings, see Section 9.21.7, "Label" on page 9-49.
9. Use the **Layout** section to modify the location of the new text on the screen. For more information, see Section 9.21.10, "Layout" on page 9-52.

> **Note:** The recommended method of modifying the layout of a UI screen is to use the UI Editor. This tool enables you to drag and drop UI controls using the mouse. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

6.3.4.4 Adding Buttons to a Screen

Add buttons to any UI screen to enable end users to perform a specific action at runtime. For example, you might create a button to add a Component to the configuration, navigate to a specific screen, or update item prices. You can also perform custom actions by associating a button with a Functional Companion. See Section 5.14, "Functional Companions" on page 5-67.

Note that although it is easy to create a button to navigate to a Component in the runtime UI, the action may fail when the end user selects it during a configuration

session. This can happen when a rule removes the Component based on a previous end user selection. When this occurs, Oracle Configurator Developer displays a message that the Component is no longer available within the current configuration session.

To add a button to a UI screen:

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface Tree and select the UI node.
3. Create or select the screen node you want to customize.
4. Click the right mouse button and select **New Button** from the menu, or choose **Create > New Button**.
5. In the Attributes View in the right pane, enter a name for the button in the field provided.
6. Open the **Description** section if it is closed and enter descriptive text about the button.
7. In the **Definition** section:
   - Choose a **Button Style** of **Rounded Both Sides**, **Rounded Left**, **Rounded Right**, or **Image**. The first three settings make the button's shape more indicative of its action. For example, you may want to choose **Rounded Right** if the button action will be to go to the next UI screen in the Model.

     Choose **Image** to display a picture instead of a button. At runtime, an end user can click on the image using the mouse to perform the **Action** that you specify. For example, you may want end users to be able to click on a picture of a configurable component so they can quickly navigate to that UI screen.
   - If you chose a **Button Style** of **Image**, click the dialog **(...)** button in the **Reference** field and then choose a graphic file from the standard Windows Open dialog. The default graphics files are stored in the %ORACLE_HOME%\Osp\Shared\ActiveMedia\ directory on the machine running Configurator Developer.

> **Note:** You can optionally use your own graphics files and store them wherever you want when designing and unit testing a UI. These graphics appear in the UI Editor to help you customize the appearance of each UI screen. To appear in the runtime UI, however, image files must also exist in the OC Servlet directory specified by OA_MEDIA. For more information, see the *Oracle Configurator Implementation Guide*.

- Enter text that describes the action of the button in the **ALT Text** field. This text appears when an end user places the cursor over the button in the runtime Configurator.
- Use the **Action** list to select the action you want to the Oracle runtime Configurator to take when an end user clicks the button. For a description of each action, see Section 9.21.6.7, "Action" on page 9-46.

8. Open the **Label** section if it is closed, then:
   - Enter a label for the button in the **Text** field. This text appears on the button in the runtime UI.
   - To use a different font for the button text, deselect the **Use Default** box, then click the dialog **(...)** button to open the Font dialog. Select a font and its related attributes, then click **OK**.
9. Use the **Layout** section to modify the location of the button on the screen and specify a **Tab Order**. For more information, see Section 9.21.10, "Layout" on page 9-52.

> **Note:** The recommended method of modifying the layout of a UI screen is to use the UI Editor. This tool enables you to drag and drop UI controls using the mouse. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

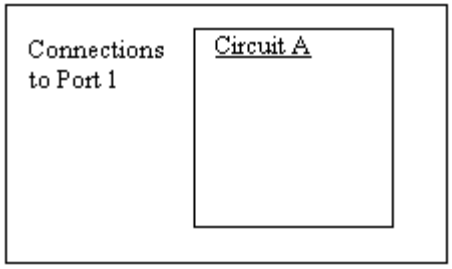
6.3.4.5 Adding a Connections Listbox to a Screen

You may want to add a **Connections Listbox** to a UI screen to display all component instances that are connected to the Model instance the end user is currently viewing. Additionally, each component displays as a **text link** that the end user can click to navigate to the component's UI screen. Connections Listboxes display the UI caption of the component on which you created the Connector (this may be different from the node's name that appears in the Model structure).

For example, you create a Connections Listbox on the UI screen node for the Model called Port 1. At runtime, the end user connects an instance of a Model called Circuit A to Port 1. When you view the UI screen for Port 1, a Connections Listbox displays Circuit A as a text link that the end user can click to navigate to Circuit A's UI screen.

Figure 6–8 shows the Connections Listbox described in this example.

Figure 6–8 A Connections Listbox in the Runtime User Interface



You typically add a Connections Listbox to a *Model's* UI screen, since only a Model can be the target of a **Connector**. However, it is also possible to add a Connections Listbox to a Component's UI screen. A Connections Listbox on a Component UI screen displays any connections to the Component's *parent* (that is, its parent Model instance); this is because an Oracle Configurator user can create connections only to an Model instance, not to a Component instance. For more information about Connectors and connectivity in general, see Section 3.6 on page 3-20.

Additionally, if the listbox appears on the UI screen of a *nested* Component (that is, a Component within a Component), it will show all components connected to the Component's parent Model. For example, Model M1 contains Component A which contains Component B. A Connections Listbox on Component B's UI screen shows all connections to Model M1.

For more information about Connectors, see Section 4.3.5, "Creating a Connector" on page 4-6.

To create a Connections Listbox:

1. In the UI module, select a UI Screen node.
2. Click the right mouse button and select **New Connections Listbox** from the menu, or choose **Create > New Connections Listbox**.

3. In the Attributes View, modify the name of the node in the field provided (optional). This text appears only in Configurator Developer.
4. Expand the **Description** section if it is closed, and enter a description of the node (optional).
5. In the Definition section:
   - To choose a different **Font** and text size for the names of Model instances that appear in the Connections Listbox at runtime, deselect the **Use Default** box, then click the list of values button (...). For more information, see Section 9.21.6.3, "Font" on page 9-45.
   - To display the path of all Models that are connected to the selected Model in the Connections Listbox at runtime, select the **Display Path in Connections List** box. For example, the path for a Model called PortA might be Model1 : NodeX : PortA. Displaying a Model's full path helps end users complete the configuration more quickly and may reduce validation errors. (Figure 6–8, "A Connections Listbox in the Runtime User Interface" on page 6-31 does *not* display the connected Model's full path.)

     Note: When you select **Display Path in Connections List**, information in the Connections Listbox does not include the *root node* of the configuration model. So, if the connected component is directly beneath the root node, the Model name appears in the Connections Listbox, but the root node name does not. In other words, the connected component must be at least 2 levels deep (nested) within the configuration model for path information to appear as described above.

6. Expand the **Label** attribute if it is closed, then enter (in the **Text** field) the text you want to use to label the Connections Listbox at runtime. This is the node's **UI caption**. (See Section 6.1.7.1, "User Interface Captions" on page 6-14.)
7. Use the following settings in the **Label** section to modify the runtime appearance and behavior of the Connections Listbox:
   - **Font** (See Section 9.21.6.3, "Font" on page 9-45)
   - **Background Color**: This setting applies only if you set **Background Style** to Opaque. (See Section 9.21.6.6, "Background Color" on page 9-46.)
   - **Background Style**

- **Align**
- **Action**: Use this setting to assign an action to the Connections Listbox's UI caption. (See Section 6.3.6, "Assigning an Action to Text" on page 6-39.)

For more information about these settings, see Section 9.21.7, "Label" on page 9-49.

8. Use the **Layout** section to modify the location of the Connections Listbox on the screen and specify a **Tab Order**. For more information, see Section 9.21.10, "Layout" on page 9-52.

Note: The recommended method of modifying the layout of a UI screen is to use the UI Editor. This tool enables you to drag and drop UI controls using the mouse. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

6.3.4.6 Sorting Feature Options

The Option Sorting settings determine the order in which Feature Options appear at runtime. The default sorting method is set at the UI level, but you can optionally override these settings for specific Features. For information about default UI settings, see Section 6.3.1, "Customizing the User Interface Default Settings" on page 6-21.

To specify an Option Sorting method for individual Feature Options:

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface Tree, then expand the *Model Name* UI node.
3. Select a Feature UI node.
4. Expand the **Option Sorting** attribute.
5. Choose a **Sort Method**. For more information about each method, see Section 6.3.1.1, "Modifying Default UI Settings" on page 6-22.

   If you choose By Property Value, select a **Property** from the list. This list displays Properties that are common to each Option of the selected Feature (at least one Property must be common to all Options of the selected Feature to use this method).
6. Choose whether to **List Selectable Options First** or choose Use Default to accept the default setting. For more information, see Section 6.3.1.1, "Modifying Default UI Settings" on page 6-22.

6.3.4.7 Customizing Features, Totals, and Resources

When you generate a new UI, Configurator Developer automatically creates a User Interface node that corresponds to each Feature you want to display in your Model. You can make your User Interface more interactive by customizing the Feature nodes in your User Interface to display as questions for the selection of options. For example, instead of displaying a list of color options with a color label, you can display “What color would you like?” next to the selection list.

You may also want to provide descriptive text for Totals and Resources to identify their purpose at runtime.

To change Feature, Total, and Resource UI nodes:

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface Tree and select the UI node.
3. Expand the UI Screen node containing the object you want to customize.
4. Select the node you want to customize. (Note that the **Description** section is read-only for both BOM and non-BOM nodes. To modify a node’s description, go to the Model module, select the node, edit the description, and then refresh the UI.)
5. Expand the **Definition** attribute if it is closed.
   - Select the type of control mechanism you want displayed for this Feature (dropdown or selection list) in the **Control** field.
   - The **Format String** field for Totals and Resources is not currently implemented.
   - To choose a different **Font** and text size for the text used to label the node at runtime, deselect the **Use Default** box, then click the list of values button (...). For more information, see Section 9.21.6.3, "Font" on page 9-45.
6. Expand the **Label** attribute, then enter the text you want to appear as the field label in the **Text** field. This is the node’s *UI caption*. For more information, see Section 6.1.7.1, "User Interface Captions" on page 6-14.
7. Modify the following settings for the node’s UI caption (label) as required:
   - **Font** (See Section 9.21.6.3, "Font" on page 9-45)
   - **Background Color**: This setting applies only if you set **Background Style** to Opaque. (See Section 9.21.6.6, "Background Color" on page 9-46)
   - **Background Style**

- **Action**: Use this setting to assign an action to the selected object's UI caption. (See Section 6.3.6, "Assigning an Action to Text" on page 6-39.

8. If the Feature is a List of Options, expand the **Option Display** attribute and select the corresponding radio button to indicate whether you want to **Label each option** or **Display pictures**. See Section 9.21.8, "Option Display" on page 9-50.
9. Use the **Layout** section to modify the location of the node on the screen and specify a **Tab Order**. For more information, see Section 9.21.10, "Layout" on page 9-52.

Note: The recommended method of modifying the layout of a UI screen is to use the UI Editor. This tool enables you to drag and drop UI controls using the mouse. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

6.3.4.8 Customizing a Connector UI Control

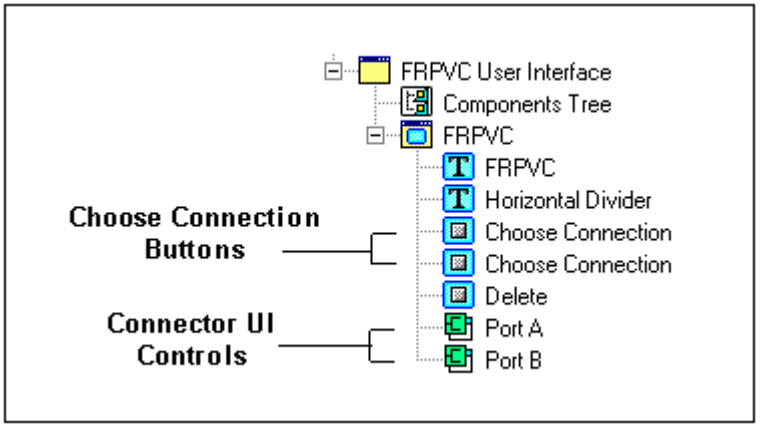
When you create a new User Interface, Configurator Developer generates a Connector UI control for each Connector in your Model. The default name of this UI control is *Target Model Name* Connector, and it appears in the Connector's parent UI screen. Modify the Connector UI control in Configurator Developer if you want to:

- Customize the runtime appearance of the target Model's name as it appears when an end user clicks a **Choose Connection** button (including text, font, size, and alignment)
- Assign an action to the Connector's UI caption

For more information, see Section 3.6, "Connectivity and Networks" on page 3-20.

Figure 6–9 shows the UI nodes for Connectors and **Choose Connection** buttons.

Figure 6–9 Connector UI Controls and Choose Connection Buttons



To modify a Connector UI control:

1. In the UI module, expand the UI Screen node containing the Connector, then select the Connector to modify.
2. Optionally modify the Connector's name in the field provided. This text appears only in Configurator Developer.

   The node description is derived from the target Model and is read-only.
3. In the **Definition** section:
   - To choose a different **Font** and size for the text used to display the target Model instance name at runtime, deselect the **Use Default** box, and then click the list of values button (...). See Section 9.21.6.3, "Font" on page 9-45.
   - To display the full path of the available target Model instances at runtime when an Oracle Configurator user clicks a **Choose Connection** button, select the **Display Path in Connection Chooser** box. For example, the full path for a Model called PortA might be Model1 : NodeX : PortA. Displaying the full path of a target instance helps end users determine which one to connect and can reduce validation errors.
4. Expand the **Label** attribute, then enter (in the **Text** field) the text you want to use to label the Connector at runtime. This is the node's **UI caption**. For more information, see Section 6.1.7.1, "User Interface Captions" on page 6-14.
5. Use the following settings in the **Label** attribute to modify the runtime appearance and behavior of the Connector's UI caption:

- **Font** (See Section 9.21.6.3, "Font" on page 9-45)
- **Background Color**: This setting applies only if you set **Background Style** to Opaque. (See Section 9.21.6.6, "Background Color" on page 9-46.)
- **Background Style**
- **Align**
- **Action**: Use this setting to assign an action to the selected object's UI caption. (See Section 6.3.6, "Assigning an Action to Text" on page 6-39.)

For more information about these settings, see Section 9.21.7, "Label" on page 9-49.

6. Use the **Layout** section to modify the location of the Connector on the screen and specify a **Tab Order**. See Section 9.21.10, "Layout" on page 9-52.

Note: The recommended method of modifying the layout of a UI screen is to use the UI Editor. This tool enables you to drag and drop UI controls using the mouse. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

6.3.4.9 Customizing a Choose Connection Button

When you create a new User Interface, Configurator Developer generates a **Choose Connection** button for each Connector node in your Model. This button appears in the Connector's parent UI screen and enables an Oracle Configurator user to connect the current component to another Model instance at runtime. If a connection is required, a red asterisk appears next to the **Choose Connection** button at runtime. For general information about connecting Model components, see Section 3.6, "Connectivity and Networks" on page 3-20.

By default, a Connector's target is the Model you specified when creating the Connector. (See Section 4.3.5, "Creating a Connector" on page 4-6.) However, if a node contains multiple Connectors, you can optionally select a different Connector from the **Target** list. This list contains only names of Connectors within the currently selected UI screen.

In the UI module you can modify a **Choose Connection** button's appearance and location on a screen. You can optionally delete the button if it is not needed.

To modify a Choose Connection button:

1. In the UI module, expand the UI Screen node containing the button you want to modify, then select the Choose Connection button.

See Figure 6–9, "Connector UI Controls and Choose Connection Buttons" on page 6-36.

2. Modify the button's attributes as desired. Each attribute (except **Target**) is described in Section 6.3.4.4, "Adding Buttons to a Screen" on page 6-28.

   If more than one Connector exists in the selected UI screen, the **Target** list displays all of the corresponding Connector node names. In this case, you can optionally select a different Connector from the list.

Note: The **Target** list displays the name of the Connector *UI control*, not the name of the target Model or the Connector node itself.

6.3.5 Hiding Objects in the Runtime User Interface

When creating a Model there may be nodes that you know should not be displayed at runtime. To prevent a node from appearing in the UI, deselect the **Display in User Interface** box in the Model module (see Section 6.1.5, "Controlling Visibility" on page 6-10). It is also possible to hide nodes at runtime by assigning an effective date or one or more **Usages** in Configurator Developer, but this method is based on variable criteria that is passed from the hosting application when a configuration session begins. See Section 3.3, "Effectivity" on page 3-4.

During unit testing a UI you may notice additional items that you do not want the end user to see. In the User Interface module, each UI node that corresponds to a Model node has a **Go To** button that allows you to navigate directly to that node in the Model Tree. This button provides an easy method of locating a specific node in the Model structure using its associated UI node.

The procedure in this section describes how to:

- Find nodes that you realize should be hidden only after generating and testing the runtime UI
- Hide the node by updating the Visibility attribute

To prevent a node from appearing in the runtime UI:

1. Select the **UI** button on the main toolbar.
2. Expand the User Interface Tree and select the UI node.
3. Select the node you want to hide.
4. At the top of the Attributes View, click the **Go To** button.

The Model Tree opens, automatically selecting the Model node corresponding to the User Interface node. The Attributes View displays a list of the User Interface nodes that are associated with the Model node.

5. Switch to the Model module by clicking the **Model** button on the Toolbar.
6. Open the **Visibility** attribute, then deselect the check box labeled **Display in User Interface**.
7. Switch to the User Interface module by clicking the **UI** button on the Toolbar.
8. Select the top-level UI node, then choose **Edit > Refresh**.
9. Expand the User Interface, and notice that the UI node associated with the hidden node no longer appears.

6.3.6 Assigning an Action to Text

In addition to assigning actions to buttons and pictures, you can assign an action to any text that appears in the runtime UI. For example, you may want to enable the end user to review additional information about specific Options of a Feature by clicking on the Feature name.

At runtime, text that has an associated action is displayed as an HTML hypertext link. As with any hypertext link, the browser window settings determine specific text formatting, such as underlining and color, which indicates that the end user can perform an action by clicking on it. (When previewing a UI screen in the UI Editor, however, text that is associated with an action does not include this formatting.)

Text objects to which you can assign an action include:

- Text that you create after generating the UI
- Page Titles (Configurator Developer automatically creates this text when you generate a new UI)
- Feature labels
- Totals and Resource labels (includes Tagged Display and Value Display nodes)

You can also assign actions to the UI captions of:

- Feature Options that appear in a selection list
- individual BOM controls (for example, BOM Models, BOM Option Classes, and BOM Standard Items)

The list of available actions depends on the type of UI node selected. You can assign any action to a Text object, such as the Page Title that Configurator Developer

creates automatically when generating a new UI and any text that you add to a screen (see Section 6.3.4.3, "Adding Text to a Screen" on page 6-27).

Features, Totals, Resources, BOM Option Classes, and BOM Standard Items can have an action of either Launch URL or Functional Companion Output. Each action is described in more detail in Section 9.21.6.7, "Action" on page 9-46.

To assign an action to a text object, Feature, Total, or Resource:

1. Go to the UI module.
2. Select the UI node to which you want to assign an action.
3. If the selected node is a Feature, expand the Definition attribute, then set the **Control** type to Selection List.
4. Expand the **Label** attribute.
5. If required, modify the selected node's UI caption by editing information in the **Text** field.
6. Select an **Action** from the list and specify **Reference** information if required. For more information, see Section 9.21.6.7, "Action" on page 9-46.

6.3.6.1 Assigning an Action to Feature Options

Feature Options do not have a corresponding UI screen node that appears in the UI module. Therefore, unless you choose the Launch URL action and use a Property to indicate the URL to open, the same action is invoked regardless of which Option the end user clicks. For more information, see Section 9.21.6.7, "Action" on page 9-46.

To assign an action to Feature Options:

1. Go to the UI module.
2. Expand the Feature UI node to modify.
3. Expand the Definition attribute and set the **Control** type to Selection List.
4. Expand the Option Display attribute and indicate whether you want to **Label each option** or **Display pictures**. See Section 9.21.8, "Option Display" on page 9-50.
5. Choose an **Action** from the list.
6. Based on the action you specified, enter a Reference, URL, or select a Property. For more information, see Section 9.21.6.7, "Action" on page 9-46.

6.3.6.2 Assigning an Action to a BOM Option Class

To assign an action to a BOM Option Class:

1. In the Configurator Developer Repository window, select the BOM Model to which the Option Class belongs and choose **File > Open**. The BOM Model opens in the Model window.
2. Go to the UI module.
3. Expand the UI screen node that contains BOM Option Class UI node.
4. Select the UI node to which you want to assign an action (for example, the Page Title or a Text node that you created).
5. Expand the **Label** attribute, then select an **Action** from the list. For a description of each action, see Section 9.21.6.7, "Action" on page 9-46.

6.3.6.3 Assigning an Action to a BOM Standard Item

To assign an action to a BOM Standard Item:

1. In the Configurator Developer Repository window, select the BOM Model to which the Item belongs and choose **File > Open**. The BOM Model opens in the Model window.
2. Go to the UI module.
3. Expand the UI screen node that contains the BOM Standard Item UI node.
4. Select the Item.
5. Expand the **Label** attribute, then select an **Action** from the list. For a description of each action, see Section 9.21.6.7, "Action" on page 9-46.

6.4 Previewing Screens with the UI Editor

Use the UI Editor to edit the layout of each User Interface object on a screen, such as the position of a button or image. Because some buttons or UI controls may overlap when they are initially created in Configurator Developer, it is recommended that you preview the layout of each screen before unit testing the Model and UI at runtime. To open the UI Editor, select a Screen node in the User Interface Tree and then choose **View > Preview**, or right-click using the mouse and then choose **Preview**.

You can open multiple UI Editor windows at the same time, but only one window can be open per UI screen. Moving or resizing a User Interface object in the UI

Editor updates the coordinate values for the selected User Interface node in the **Layout** section in the Attributes View. Similarly, any changes you make in the **Layout** section are reflected in the UI Editor. See Section 9.21.10, "Layout" on page 9-52.

You can use the UI Editor to cut, copy, paste, and delete objects. You can also select multiple objects (by holding the Shift key and clicking with the mouse) to control alignment, resize objects, and modify the horizontal and vertical spacing between each object.

The UI Editor does not show *all* aspects of a UI screen's appearance at runtime. For example, the UI Editor does not show the runtime Navigation Tree, default page banner, or the **Summary**, **Availability**, **Done**, and **Cancel** buttons that appear by default in the DHTML UI.

7

Testing and Debugging

After you create Model structure, configuration rules, and a UI, you need to test and debug your Model before making it available for use in a production environment. This chapter describes how to test each part of your Model using the Configurator Developer Test / Debug module.

7.1 The Test/Debug Module

Use the Test / Debug module to review a portion of the Model structure, view the Model structure in a UI before creating any configuration rules, or as a final check before releasing the Model for testing by external users. This module enables you to review and make iterative changes to a Model while it is still in development. When a configuration model is complete and you want to make it available to others for testing or production activities, you must create a new **publication**. See Section 2.2, "Publishing" on page 2-11.

Before testing your runtime Oracle Configurator using the Test / Debug module, you must:

- Create Model structure and a User Interface (see Section 4.1, "The Model Window" on page 4-1)
- Generate the Active Model (see Section 5.1.1, "Compiling Rules" on page 5-2)
- Select the deployment environment for testing your runtime Oracle Configurator (see Section 9.9.2.1, "Selecting a Test Environment" on page 9-15)

If you changed the Model since the last time the User Interface was generated, you must refresh the UI or create a new one prior to testing. If you changed the Model or any configuration rules, you must also regenerate the Active Model.

When you are ready to test your Configurator, click the **Test** button the main toolbar, or select **View > Test/Debug**. When you do this, you can view the Model

based on the effectivity that you defined for each node in Configurator Developer. See Section 7.1.1, "Testing your Model" on page 7-2.

When you click **OK**, Oracle Configurator Developer generates and displays an runtime Oracle Configurator window created from the Model and configuration rules you defined and the effectivity parameters that you specified. This testing environment is called the **Active User Interface**.

Testing Environments

The environment in which Oracle Configurator Developer displays the Active User Interface depends on the settings in the Options dialog. See Section 9.9.2, "Options" on page 9-14.

If you are testing your Model in the Dynamic HTML in a browser or Java Applet test environment:

1. Configurator Developer sends an initialization message to the UI servlet, using the URL you specified in the Options window (**Tools > Options**).
2. Your current session information (database instance, username, password, etc.) is used to connect to the Oracle Configurator schema.
3. If you defined more than one User Interface in Configurator Developer, select the one you want to test from the popup menu.
4. Your default Web browser or a Java applet opens, displaying the current Model using the User Interface you specified.
5. Configurator Developer writes the contents of the initialization message to a file in the directory specified by your computer's TMP user environment variable (for example, C:\WINNT\Temp). If your testing environment is a Java applet, this file is called applethtm.htm. Otherwise, the file is called dhtmlhtm.htm. You can view this file using a standard text editor.

7.1.1 Testing your Model

Verify that the Model structure as displayed through the User Interface functions effectively in the runtime Oracle Configurator. To do this, you must first select the deployment environment by defining parameters in the Options window. See Section 9.9.2.1, "Selecting a Test Environment" on page 9-15.

View and test the structure of your Model for visibility and other Properties that you have applied. See Section 7.1.4, "Configuring an Item in a Runtime Oracle Configurator" on page 7-3 for details about how the User Interface works.

7.1.2 Testing your Configuration Rules

Test configuration rules to determine they produce the desired results in the runtime UI. It is good practice to test configuration rules incrementally. If you receive an error in the runtime Oracle Configurator, you can temporarily disable specific rules to determine which rule is causing the error, and then modify its definition to resolve the problem. For more information, see Section 5.4.2 on page 5-19.

You can create a report that lists all configuration rules for a Model by generating a Model Report. See Section 9.5 on page 9-5.

7.1.3 Testing your User Interface

Note whether the User Interface has the look and feel that you want for your runtime Oracle Configurator. Check that the screens present appropriate information to the end user in an easily usable format.

See Section 6.1.2 on page 6-2 for more information on how Model structure and User Interface are related.

7.1.4 Configuring an Item in a Runtime Oracle Configurator

You can use the following steps for any runtime Oracle Configurator:

1. Oracle Configurator contains a selection pane in which you choose options. It may also contain a Model structure pane (on the left), which displays a tree of the components in the configurable product.
2. In the selection pane, begin making selections from the options that are presented. Table 7–1 describes how to select each type of option at runtime.

Table 7–1 Selecting Options at Runtime

To select this . . .	Do this . . .
One option	Select the check box for the desired option.
One option in a group of mutually exclusive options	Select the check box for the desired option. (This list might be presented as a set of radio buttons.)
One or more options in a group that allows multiple selections	Select the check boxes for all the desired options.

Table 7–1 (Cont.) Selecting Options at Runtime

To select this . . .	Do this . . .
A quantity for a numeric option	Enter the desired quantity. Entering a quantity greater than zero automatically selects the option.

3. As you make selections, they are automatically validated against the rules that have been defined for the Model or item that you are configuring.

   If you change one of your selections, it can automatically change the choice of valid selections for other features of the product, according to the configuration rules. You see those changes when you select an affected feature.

   If you make a selection that violates a configuration rule, the Configurator window displays a message describing the violation and your options for proceeding.

4. When you are finished making selections for one component, you move on to the next components, until you configure the entire product.

   For information on navigating, see Section 6.1.2, "How the Oracle Runtime Configurator Displays the Model" on page 6-2.

5. View the summary to display all the selections you have made, and other option information, such as quantity, price, and the total price for the configured product.

6. If an **Availability** button is displayed, click it to show the Available To Promise (ATP) dates for the relevant items.

7. When you have finished the entire configuration, click **Done** to save the configuration and continue processing your order.

   If the configuration is satisfied, you have made all the required valid selections.

   If the configuration is unsatisfied, a message tells you how to either ignore it and continue, or return to the selection pane to finish the configuration.

   The host application then closes the Oracle Configurator window and continues processing your order.

> **Note:** Ending a configuration session using a method other than clicking **Done** causes all selections to be lost (for example, clicking **Cancel** or choosing **File > Close** from the browser window).

7.1.5 Viewing Changes to the Model in the Runtime User Interface

As you test the Model, you may notice parts of the UI that you want to change or rules that do not function as you intended. Use Configurator Developer to make any necessary changes. However, you will notice that the Dynamic HTML in a browser and Java Applet testing environments do not immediately reflect changes you make.

For example, you are testing your configuration model and notice an error in a button caption, so you fix the caption in Developer. The change you make does not immediately appear in the Model currently running in your browser. To view recent configuration rules or UI changes, you must open a new configuration session in a new browser window by clicking the **Test** button in the main toolbar.

7.2 Configurator Developer Messages

As you work in Configurator Developer, the system may occasionally display messages alerting you to errors or warning about potential problems in your configuration model. For example, if you create a rule but do not define it, Configurator Developer displays a warning message when you generate the Active Model. These messages, along with other execution status information, are logged by Configurator Developer in the datastore log.

You can display the contents of the datastore log in the Log Messages window by enabling the **Show Datastore Log** option. To do this, choose **Tools > Options**, select the **Log** tabbed region, and select the **Show Datastore Log** box. When you click **OK**, the Log Messages window appears.

For more information on using the Log Messages window, see Section 9.9.2.4, "Displaying the Log Messages Window" on page 9-17.

You can also specify that log messages should be written to a file by entering a directory path and file name in the Log File field. Logging execution messages to a file preserves this information and may be useful if you are having problems with Developer and need to provide information to customer support. By default, messages are written to a file called `Developer.log` in the directory `Osp\Developer` in your Oracle install directory.

Status messages also have a severity level. You can specify which messages to include in the log file by choosing **Settings > Report Settings > (severity level)** from the Log Messages window. Oracle Configurator Developer logs all messages of a severity equal to or higher than the level you select, and discards all messages in categories less severe than this level. Note that setting a very low severity level

can affect system performance, due to the large number of messages logged. The default severity level is Notification.

Following are message severity levels and their categories from the least to the most severe.

Tracing and informational messages

1. DetailTrace
2. Info

Warnings

1. Empty/EOF
2. Notification
3. Warning

Errors

1. Error
2. Data Damaged
3. FATAL

7.2.1 Error Messages

- The "Runtime Error '7' " and "out of memory" error messages are usually caused by too little virtual memory.
- If you receive the error "The name is not in use for a subkey or named value", check the TNS, ODBC, DSN, and spx.ini information. See the *Oracle Configurator Installation Guide* for more information.
- The following message is almost always the result of an inconsistency in your Active Model: "WebUI Initialization Failure", "oracle.apps.cz.logic.EngineFatalLoadFileException: Error loading file: null: unknown No such variable defined". Choose **Tools > Generate Active Model** to regenerate the Active Model.
- The following error message occurs when you create a BOM Model Tree UI for a non-BOM Model, and then try to access the UI using the Test module: "WebUI Initialization Failure: User interface is not compatible with the client type." To

resolve this, regenerate the UI and select **Components Tree UI** for the **UI Style** setting. See Section 6.2, "Generating a New User Interface" on page 6-16.

8

Using Repository Window Tools

Oracle Configurator Developer provides a Windows-based user interface you use to organize and maintain configuration Models and the attributes they share. This chapter discusses the tools you can use to store and manage configuration Models, Effectivity Sets, and Usages.

8.1 Oracle Configurator Developer Windows

Oracle Configurator Developer consists of a Model window and a Repository window. The **Model window** and its subgroup of modules enable you to build Model structure, define configuration rules, and create and customize a User Interface to your configurator. See Section 9, "Using Model Window Tools" on page 9-1.

This chapter describes the tools available in the **Repository window**. For an overview of using Oracle Configurator Developer tools to build a configuration model, see Section 1.4.3, "Building Models" on page 1-7.

8.2 Repository Window

The Repository window is the first window that appears when you log in to Oracle Configurator Developer. In this area you organize and maintain Models and their shared attributes in a single location, so changes you make once affect all elements assigned to that attribute. You also use the Repository window to publish Models for production, testing, or other activities. For more information about publishing, see Section 2.2 on page 2-11.

When you select an existing Model node and choose **File > Open** or double click in the Repository window, the Model opens in the Model window so you can modify its structure, configuration rules, and User Interface. The Repository and Model

windows can be open at the same time so you can view and manage Repository entities while modifying an existing Model.

Note that the Model name and description that appears in the Repository are not necessarily the same as the name and description of the root node in the Model window. You can enter unique Model names and descriptions in the Repository to make it easier to organize and find Models when you want to edit them. When you open a Model for editing, Configurator Developer displays the Model name that appears in the Repository window in the title bar of the Model window.

Use the Repository to:

- Create, organize, and maintain Model nodes
- Create, organize, and maintain shared Model attributes including Effectivity Sets and Usages
- Create Folders to store and organize Model nodes and Model attributes in a logical manner
- Publish configuration models
- Delete Effectivity Sets, Usages, and Models

8.2.1 Repository Entities

Entities that you create and maintain in the Repository include:

- Model nodes
- Effectivity Sets
- Usages
- Folders

Effectivity Sets and Usages are attributes you use to control the availability of Model nodes and these attributes can be shared by multiple configuration models and Model elements simultaneously. See Section 3.3, "Effectivity" on page 3-4.

Repository Folders are similar to a file directory system. You use Folders to store and organize configuration Models, Effectivity Sets, Usages, and other Folders (subfolders). See Section 8.13, "Tree Views" on page 8-12.

When you create a Model, Effectivity Set or Usage in the Model window, Configurator automatically creates the entity at the top level in the Repository. You can then move the new entity to a specific Folder using cut and paste.

8.3 Managing Repository Entities

This section describes the operations you can perform on Repository entities, including Models, Folders, Usages, and Effectivity Sets.

8.3.1 Creating

In the Repository, you can create new Effectivity Sets, Models, and Usages by selecting a Folder, and then:

- Choosing the appropriate action from the **Create** menu

  or

- Clicking the right mouse button and selecting the appropriate action (such as **New Effectivity Set**)

You can also create these entities by copying existing Effectivity Sets, Models, and Usages. When you use copy and paste, Configurator Developer makes a copy of the selected entity and gives the new copy a unique name. For example, if you copy Effectivity Set ”EffectivitySet1," the copied set is called "Copy of EffectivitySet1." If this name exists, then the new name has a number added to the name. For example: Copy(n) of Effectivity Set1. See Section 8.9, "Create Menu" on page 8-8.

8.3.2 Moving

Use the Cut and Paste commands to move an entity from one location in the Repository to another. These commands are available from the **Edit** menu and by right clicking using the mouse.

8.3.3 Modifying

You can modify Effectivity Sets and Usages in both the Repository and the Model windows. You can modify the name and description of an Effectivity Set at any time, but you cannot modify the name of a Usage if it is assigned to an existing publication.

Configurator Developer does not display an error or warning message when you edit an entity’s name or description because this change does not adversely affect the availability of Model components. The change simply propagates to all elements that share that entity.

However, if you modify the *date range* of an Effectivity Set that is shared by more than one Model component, Configurator displays a warning message. This is

because modifying effective dates can affect multiple components across many different configuration Models. The warning that Configurator Developer displays in this case is similar to the following:

"This change will affect 2 Model nodes and 3 rules, of which 2 are members of a Rule Sequence. Do you wish to continue?"

At this point you can do one of the following:

- Click **OK** to make the change.
- Click **No** to cancel the changes and return to the Edit Effectivity Set window. You can then click the **Member** button to view all of the Model components to which this Effectivity Set is assigned. To find out more information about a Model component, select it and click **Go To**. Once you know which components will be affected, you can decide whether to proceed with the change.

  To change a Model's structure, rules, or UI, you must open it in the Model window. To do this, select the Model node in the Repository, then either double click using the mouse, or choose **File > Open**. See Section 4, "Constructing Model Structure" on page 4-1.

8.3.3.1 Updating Multiple Effectivity Sets

You can update multiple Effectivity Sets at the same time by choosing **Tools > Mass Edit Effectivity Sets** in the Repository window. When you do this, the Select Effectivity Sets for Mass Edit window displays your folders and Effectivity Sets alphabetically by name.

To view the Effectivity Sets in a different order, choose **Sort by** parameters. You can also modify the sort by specifying secondary or tertiary sort parameters, including Name, Description, Start and End date, and then specify either ascending or descending order. After specifying sort options, click **Sort**.

Select the Effectivity Sets that you want to edit, then click **Edit**. In the **Edit Effectivity Dates** window, choose whether you want the selected Effectivity Sets to be **Always Effective**, **Never Effective**, or select **Change** to modify the start and end dates. If you select **Change**, a dropdown list displays the following options:

- Start Date only
- End Date only
- Start and End Date

Select the **Ignore Warnings** check box if you do not want to view warnings about how the selected Effectivity Sets are used.

Click **OK** to validate the new settings. Configurator Developer reports all errors and warnings in a separate dialog. If there are any errors, Configurator Developer does not make the changes and you must modify the dates before proceeding. If there are warnings, click **OK** if you want to make the changes anyway; otherwise, click **Cancel**.

Changing the dates on an Effectivity Set maintains the dates defined on rules in any Rule Sequences. Therefore, Configurator Developer lists the changes that will occur if the changes are made and allows you to either continue or cancel the operation. If the new dates cause any errors in a Rule Sequence, Configurator Developer does not update the selected Effectivity Sets; in this case, you must enter different Start or End dates.

8.3.4 Renaming

To change the name of a Model, Folder, Effectivity Set, or Usage, select it and then choose **Edit > Rename**.

8.3.5 Deleting

You can delete a Model only if it is not referenced by another Model and no publication exists for that Model. To see if a Model is referenced by one or more other Models, select it, then choose **View > References**. For more information, see Section 2.1, "Model Referencing" on page 2-1. Configurator Developer displays a message if you attempt to delete a Model for which a publication exists. See Section 2.2, "Publishing" on page 2-11.

You can delete an Effectivity Set or Usage if it is not assigned to any Model components. To delete an Effectivity Set or Usage that is in use, you must first delete the assignment from the Model window. To see all components to which an entity is assigned, select it and then choose **View > Members**. In the Members dialog window, click **Go To** to view the entity in the Model window. In the Model window, delete the association between the Model component and the entity by modifying entries in either the **Member of Effectivity Set** field or the **Usages** field.

You can delete a Folder only if it is empty.

8.4 Repository Editing Tools

Each menu on the Oracle Configurator Developer Repository window menu bar contains commands for performing a set of related operations.

The menus in Configurator Developer are context-sensitive. The set of commands on a menu changes to reflect the context in which it is being used. An example is the Create menu. When you change to a different Oracle Configurator Developer module, the set of things that you can create changes.

The commands on Oracle Configurator Developer menus are also available on context-sensitive pop-up menus. When you select a node in one of the panes on the left-hand side of the Oracle Configurator Developer window, then click the right-hand mouse button, a menu appears that lists appropriate commands for the operations you perform on the selected node.

The Repository window provides the following menus:

- File Menu on page 8-6
- Edit Menu on page 8-7
- Create Menu on page 8-8
- View Menu on page 8-9
- Tools Menu on page 8-10
- Help Menu on page 8-12

8.5 File Menu

Use the **File** menu to open a Model, Effectivity Set, or Usage for editing, modify Developer preferences, or exit Configurator Developer. Table 8–1 describes each command on the **File** menu.

Table 8–1 Repository Window FIle Menu Commands

Command	Description
Open	Opens the selected entity for editing. Models open in the Model window. Usages and Effectivity Sets open in an Edit window.
Properties	Displays the creation and last modified dates for the selected entity. You can also use this option to modify the description of a Model node.
Exit	Exits the Oracle Configurator Developer application.

8.6 Edit Menu

Use this menu to perform editing operations on the currently selected node or value. Table 8–2 describes each command on the **Edit** menu.

Table 8–2 Repository Window Edit Menu Commands

Command	Description
Cut	Cuts the selected node or value. A selected node is displayed in a disabled state until you Paste it.
Copy	Copies the selected node of any type or value.
Paste	Pastes the cut or copied node or value.
Delete	Prompts you to confirm the deletion, then deletes the selected node. See "Deleting" on page 8-5.
Rename	Renames the selected node.

You can also use the buttons on the Repository Editing Toolbar for some of these commands.

8.7 Repository Window Keyboard Shortcuts

You can access the cut, copy, and paste commands in the Repository window using keyboard shortcuts. Table 8–3 lists each command on the Repository **Edit** menu and the corresponding keyboard shortcut for each.

Table 8–3 Repository Window Keyboard Shortcuts

Command	Shortcut
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V

8.8 Repository Editing Toolbar

Use this toolbar to perform basic editing operations on the selected Model node or entity.

Icon	Command
	Open
	Cut
	Copy
	Paste

8.9 Create Menu

Use this menu to create new Model nodes, Effectivity Sets, or Usages. Whether you can create a node or an entity depends on the current selection. For example, if a Folder is selected, you can create a Model, Folder, Effectivity Set, or Usage.

Table 8–4 describes each command on the **Create** menu.

Table 8–4 Repository Window Create Menu

Command	Description
Model	Creates a new Model in the currently selected Folder. See Section 8.9.1, "Creating Models" on page 8-9.
Effectivity Set	Creates a new Effectivity Set in the currently selected Folder. See Section 3.3.1, "Effectivity Sets" on page 3-5.
Usage	Creates a new Usage in the currently selected Folder. See Section 3.3.2, "Usages" on page 3-6.
Folder	Creates a subfolder in the currently selected Folder. See Section 8.9.2, "Creating Folders" on page 8-9.

8.9.1 Creating Models

To create a Model in the Repository, select a Folder, then choose **Create > Model** or right click using the mouse and choose **New Model**.

After creating a Model node in the Repository, you can open it in the Model window to define its structure and configuration rules and generate a new User Interface. You can open only one Model at a time in the Model window. See Section 4, "Constructing Model Structure" on page 4-1.

When you create a Model by building it in the Model window or by importing a Bill of Materials, Configurator automatically creates a corresponding Model node in the top level of the Repository (that is, not within a Folder). You can then move the new Model node into a Folder using the Cut and Paste commands. See Section 3.4, "Imported BOM Models" on page 3-7.

You can have more than one Model with the same name in the Repository, but they must be stored in separate Folders. Note that the Model name that appears in the Repository might not be the same as the name of the Model root node in the Model window. Providing unique, descriptive Model names in the Repository makes it easier to manage and locate Models you want to modify. When you open a Model for editing, Configurator Developer displays the name that appears in the Repository window in the title bar of the Model window. This name might be different from the name of the Model node itself.

8.9.2 Creating Folders

To create a Folder in the Repository, select the root Folder at the top of the Repository window, then choose **Create > Folder**. To create a subfolder, select an existing Folder then choose **Create > Folder**, or right click using the mouse and choose **New Folder**. For example, if a Folder TEST1 is selected and you choose **Create > Folder**, Developer creates a new subfolder in TEST1. You can also create a subfolder by cutting an existing Folder and pasting it into another Folder.

8.10 View Menu

Select one or more options on the View menu to control which entities appear in the Repository. These options include All, Model, Effectivity Sets, Usages, and View by Folder. View by Folder is an optional "toggle" setting that displays Folders in addition to the other option you select. For example:

- To view all Models and Folders in the Repository, select both **Model** and **View By Folder**

- To view all Models, Effectivity Sets, Usages, and Folders, select **All** and **View by Folder**
- To view only Models, select only **Model**
- If **View by Folder** is not selected, no Folders appear in the Repository window

The default view is **View All by Folder**.

8.11 Tools Menu

Use the **Tools** menu to open the Model Publishing window. The Model Publishing window enables you to create, update, or delete Model publications. See Section 2.2, "Publishing" on page 2-11.

8.11.1 The Model Publishing Window

Following is a description of each region and button in the Model Publishing window.

The Show Region

All published Models: Choose this radio button to view all publications on the specified database instance.

Selected Model: Choose this radio button to view only publications of the currently selected Model. The Model name is the one that appears in the Repository window, not the name of the root node in the Model window (if they are different).

Instance: Choose this list to view Model publications according to the database on which it exists.

The List of Model Publications

ID: The Model publication ID number. This number is generated by the database when the publication is created.

Model: The name of the Model as it appears in the Repository window. (The root node that appears in the Model window and the Model name in the Repository window can be the same or different, depending on how you define them.)

UI: The User Interface associated with this publication. You can create multiple UIs for a Model in Oracle Configurator Developer, but a Model publication can have only one UI. You can also create a publication without specifying a UI by choosing None for this option.

Instance: The database on which the Model data associated with this publication exists.

Published: The date the selected publication was created.

Status: The current status of the selected publication. Values include Complete, Pending, Update Pending, Processing, and Error. See the *Oracle Configurator Implementation Guide* for more information.

Note: You can modify the order in which publications appear in the Model Publishing window by clicking on a column heading. For example, click on **Model** to display the list of publications alphabetically by Model name. The first time you click on a heading, the publications appear in ascending order; click the same heading again to display them in descending order.

The Applicability Region

This region shows the applicability parameters defined for the selected Model, such as the publication mode, application, Usages, and effective dates. You set these parameters when creating a new publication or editing an existing publication. Applicability parameters are explained in Section 2.2.1.1.1 on page 2-16.

Buttons in the Model Publishing Window

New: Click this button to create a new Model publication. When you click **OK** in the New Publication window, Configurator Developer creates a publication with a status of Pending. Creating a new publication is described in Section 2.2.1.1 on page 2-14.

New Copy: Click this button to create a new Model publication based on an existing publication's applicability parameters. For example, if you want to create a new publication that will have applicability parameters that are similar to an existing publication, select the existing publication, then click **New Copy**. In the Copy Publication window, modify information for the new publication as required. See Section 2.2.1.2, "Copying an Existing Publication" on page 2-19.

Edit: Click this button to modify the applicability parameters for an existing publication. This function changes the availability of the publication, it does not actually copy Model data. Editing publications is described in Section 2.2.1.5 on page 2-22.

Republish: Click this button to update a previously published Model with any changes you made to the Model in Configurator Developer. Republishing is described in Section 2.2.1.5 on page 2-22.

Delete: Click this button to delete a publication record from the database. This function does not remove any Model data from the database. Deleting a publication is described in Section 2.2.1.6 on page 2-23.

Refresh: Click this button to update the **Status** column for the list of publications. The status of one or more publications may change when one of the publishing concurrent programs completes successfully. (For details about the publishing concurrent programs, see the *Oracle Configurator Implementation Guide*.)

Close: Click this button to close the Model Publishing window. Configurator Developer updates the list of publications when you open the Model Publishing window, so closing and opening it may update the status of one or more publications (for example, if a concurrent program has just completed).

8.12 Help Menu

Use this menu to get information about Oracle Configurator Developer and how to use it to construct your configurator.

Table 8–5 describes each command on the Help menu.

Table 8–5 Repository Window Help Menu

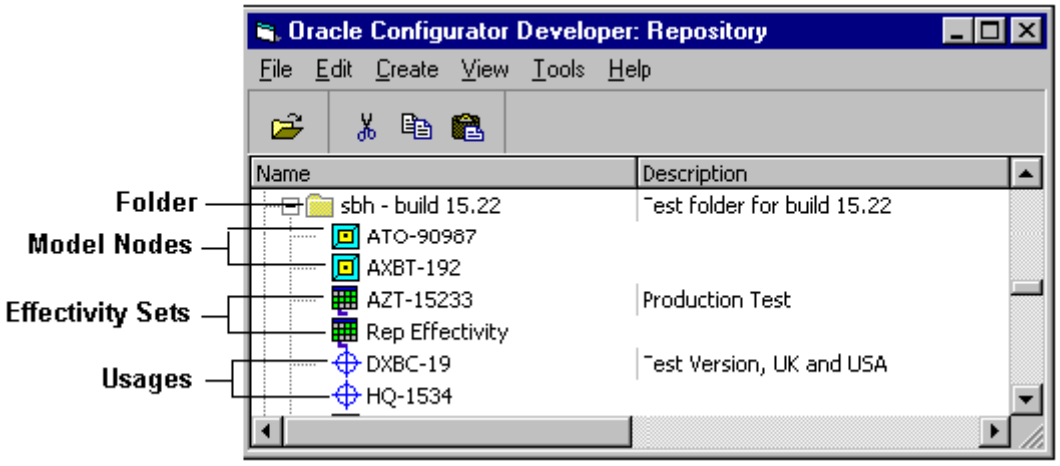
Command	Description
Help Topics	Opens the online help for Oracle Configurator Developer.
About Oracle Configurator Developer	Displays information about this version of Oracle Configurator Developer.

8.13 Tree Views

Oracle Configurator Developer displays Repository entities in hierarchical structures called **tree views**. Tree views show the location of each entity in the Repository and Folder structure contain others. You can expand or collapse branches of the tree using the plus (+) and minus (-) controls. These controls appear on any Folder that contains one or more Models, Effectivity Sets, or Usages.

When a Folder is collapsed, only the root node (the Folder) remains visible. Nodes in Oracle Configurator Developer tree views have names and graphical icons

associated with them. The graphical icons provide visual cues that identify each Repository entity as a Model, Effectivity Set, Usage, or Folder.



8.13.1 Nodes Present in a Tree View

8.13.1.1 Name

Provides a name for the selected node. When you create a new node, Developer prompts you to enter a name. You can change the name of a Model node by right clicking with the mouse and choosing Rename. To modify the name of a Usage or Effectivity Set, select the entity, then choose **File > Open**, or right click using the mouse and then choose **Open**.

Note: Enter a name that is unique, meaningful to the Oracle Configurator end user and, if possible, descriptive of the node's intended purpose.

8.13.1.2 Description

Provides a description of the selected node. When you create a new node, Developer prompts you to enter a description. You can change the description of a node at any time.

To modify the description of a Model node, select it, then choose **File > Properties**.

To modify the name of a Usage or Effectivity Set, select the entity, then choose **File > Open**, or right click using the mouse and then choose **Open**.

9

Using Model Window Tools

Oracle Configurator Developer provides a Windows-based user interface you use to build the Model structure and configuration rules that drive your completed configuration. This chapter discusses the tools available in the Configurator Developer Model window that you can use to create and maintain configuration Models.

9.1 Oracle Configurator Developer Windows

Oracle Configurator Developer consists of a Repository window and a Model window. You use the **Repository window** to manage the Oracle Configurator Developer Models on which your configurators are based. See Section 8, "Using Repository Window Tools" on page 8-1. To modify a Model's structure, rules, and User Interface, you must open it for editing in the **Model window**.

This chapter describes the tools available in the Model window. For an overview of using Oracle Configurator Developer tools to build a configuration model, see Section 1.4.3, "Building Models" on page 1-7.

9.2 The Model Window Modules

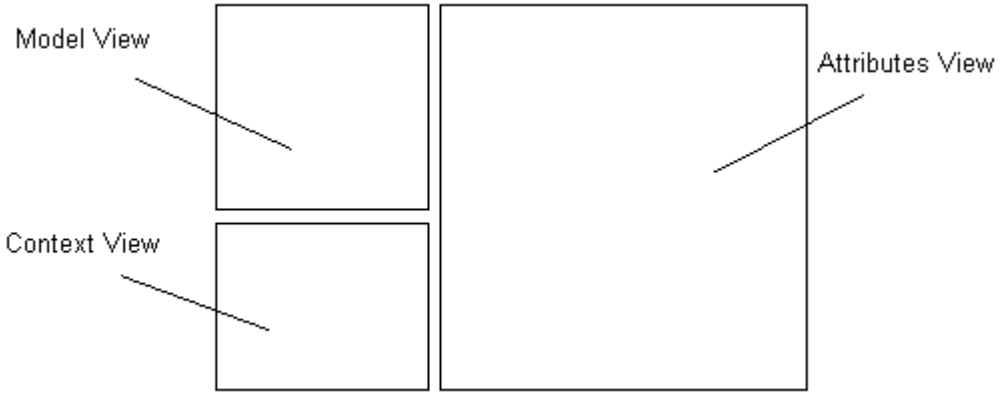
A module is a subgroup of the total suite of tools that you use to create and modify configuration Models in Oracle Configurator Developer. The Model window consists of the following modules: Model, Configuration Rules, User Interface, and Test. Each module provides tools for accomplishing a common set of tasks such as building a Model or defining rules.

Table 9–1 Model Window Modules

Module	Description
Model	Use this module to view and develop Item Master data, build Model structure, and define attributes for each node, including Visibility, Properties, and Effectivity. See Section 4.1, "The Model Window" on page 4-1.
Rules	Use this module to create the configuration rules that define constraints between elements of your Model to control how products can be configured. See Section 5.1, "The Configuration Rules Module" on page 5-1.
UI	Use this module to define one or more User Interfaces for your application and customize them for your specific company needs. The structure of the generated User Interface is derived from the structure of the Model. See Section 6.1, "The User Interface Module" on page 6-1.
Test / Debug	Use this module to view your selected test environment (either Java applet or DHTML UI) and test Model structure, rules, effectivity, and User Interface customizations. See Section 7.1, "The Test / Debug Module" on page 7-1.

9.3 The Model Window

The Model window is divided into three panes. Each pane contains a view that you use for particular tasks in creating your application.



9.3.1 Model View

The Model View appears in the upper-left pane. It is always available, since the Model is the organizing core of your configurator.

9.3.2 Context Tree View

The Context Tree View appears in the lower-left pane. It displays a different set of nodes, depending on which module you are currently working in. Table 9–2 lists each module and the information that appears in the Context Tree view when that module is active.

Table 9–2 Active Module and Context Tree View Display

Current Module	Context Tree View displayed
Model	The tree of Item Master Items and Item Types.
Configuration Rules	The tree of configuration rules for your Model.
User Interface	The tree of User Interface elements for your Model

See Section 9.11, "Tree Views" on page 9-18 for information about a specific context tree.

9.3.3 Attributes View

The Attributes View appears in the right-hand pane in the Model window.

An Attributes View is available on every node of every tree used in Oracle Configurator Developer. The Attributes available in an Attributes View depend on the selected module and node. Attributes for the Developer modules are described in the following sections:

- Section 9.14, "Model View" on page 9-21
- Section 9.15, "Item Master Attributes" on page 9-32
- Section 9.17, "Model Attributes for the Configuration Rules Module" on page 9-35
- Section 9.18, "Configuration Rules Attributes" on page 9-36
- Section 9.20, "Model Attributes for the User Interface Module" on page 9-40
- Section 9.21, "User Interface Attributes" on page 9-41

Each available Attribute is presented in a separate section. To display the contents of a section, click on the Right Arrow icon next to a section's label. When you do this, the icon changes to a down arrow, and the section (Attribute) opens. To hide the contents of a section, click on the Down Arrow icon. Table 9–3 shows the Right and Down Arrow icons.

Table 9–3 Attribute Icons

Icon	Definition
	Right arrow icon. Indicates a closed attributes view section with contents hidden.
	Down arrow icon. Indicates an open attributes view section with contents visible.

9.4 Model Window Menu Bar

Use the tools on the Oracle Configurator Developer Model window menu bar to edit information in the Model window. Each menu on the toolbar contains commands for performing a set of related operations.

The menus in Configurator Developer are context-sensitive. The set of commands on a menu changes to reflect the context in which it is being used. An example is the Create menu. When you change to a different Oracle Configurator Developer module, the set of things that you can create changes.

The commands on Oracle Configurator Developer menus are also available on context-sensitive pop-up menus. When you select a node in one of the panes on the left-hand side of the Oracle Configurator Developer window, then click the right-hand mouse button, a menu appears that lists appropriate commands for the operations you perform on the selected node.

Oracle Configurator Developer provides the following menus:

- "File Menu" on page 9-5
- "Edit Menu" on page 9-5
- "Create Menu" on page 9-9
- "Create Menu for the Item Master" on page 9-10
- "Create Menu for the Configuration Rules Module" on page 9-10
- "Create Menu for the User Interface Module" on page 9-11

9.5 File Menu

Use this menu to generate a Model Report and to close the Model window. Table 9–4 lists the commands on the **File** menu.

Table 9–4 Model Window File Menu Commands

Command	Description
Model Report	Generates a report using Microsoft Word 97 or higher. You can choose whether to include a description of the Model's structure, configuration rules, and Item Master.
Exit	Closes the Model window.

Note: Before generating a Model Report, open Microsoft Word and verify that the default "Save as type'' is either Word Document or Word 97. If the default is a different value, Configurator Developer displays an error when you generate the report.

9.6 Edit Menu

Use this menu to perform editing operations on the currently selected node or value. Table 9–5 lists the commands on the **Edit** menu.

Table 9–5 Model Window Edit Menu Commands

Command	Description
Cut	Cuts the selected node or value. A selected node is displayed in a disabled state until you Paste it.
Copy	Copies the selected node of any type or value.
Copy With Rules	Copies the selected Component node and any configuration rules contained within its structure. See Section 9.6.1.1.1, "Copy with Rules" on page 9-6.
Paste	Pastes the cut or copied node or value.

Table 9–5 (Cont.) Model Window Edit Menu Commands

Command	Description
Rename	Renames the selected node.
Delete	Prompts you to confirm the deletion, then deletes the selected node.
Refresh	Rebuilds the selected User Interface to reflect changes you have made in the Model.
Add to Sequence	Enables you to add a selected rule to Rule Sequence
Delete from Sequence	Enables you to delete a rule from a Rule Sequence
Reorder in Sequence	Enables you to Reorder rules in a Rule Sequence. See Section 5.12, "Rule Sequences" on page 5-47.
Find	Enables you to find a node in the Model or Item Master.

You can also use the buttons on the Editing toolbar for some of these commands. See Section 9.6.2, "Editing Toolbar" on page 9-7.

9.6.1 Cut, Copy, and Paste

The cut, copy, and paste operations are mechanisms for moving or reusing Model, rules, and User Interface nodes. You can cut, copy and paste in any textbox in the Attributes view.

9.6.1.1 Model and Rules Modules

You can cut, copy and paste on any node in the Model or Rules tree views. Nodes that are cut or copied in the Model module can be pasted under another node in the Model tree with the following limitations:

- Option nodes can be pasted only on a Feature node.
- Feature nodes can be pasted only on a Component or Product node.
- Component nodes can be pasted only on a Component or Product node.
- Total and Resource nodes can be pasted only on a Product or Component.
- You can cut and paste a node with Populators, but if you copy a node with Populators, the Populators are lost in the copy.

9.6.1.1.1 Copy with Rules

The **Copy with Rules** command copies the selected Component node and all of its associated configuration rules. This function copies only rules whose participants are entirely contained in the subtree of the node you are copying and places the copy of each rule in the corresponding Rule Type folder. If a rule relates nodes in the subtree with any nodes that are elsewhere in the Model (that is, not in the structure of the copied Component node), then that rule is not copied.

You may want to use **Copy with Rules** if, for example, you want constrain the copied structure the same way as the source structure, but do not want to recreate all of the rules from scratch.

9.6.1.2 User Interface Module

You can also cut and copy Picture, Text, and Button UI nodes and paste them as children of any Screen node regardless of the Component or Option Class associated with the Screen. You can also drag and drop Picture, Text, and Button nodes.

If you cut or copy a Picture or Button node that has an associated action of either Add Component or Delete Component action, and paste it to a different screen, the Reference selection in the pasted node is blank, and the **Reference** selection list contains entries appropriate to the node's new location. See Section 9.21.6.7, "Action" on page 9-46 for more information on the Reference selections available for these actions.

Cut, Copy, and Paste also function in the UI editor. They are available from the UI Editor's **Edit** menu. You can also cut, copy, and paste UI objects between the UI treeview and the UI Editor. See also, Section 9.22.2, "UI Editor Edit Menu" on page 9-53.

9.6.2 Editing Toolbar

Use this toolbar to perform basic editing operations on the selected node or text.

Icon	Command
	Cut
	Copy

Icon	Command
	Paste

9.6.3 Find

This option opens a dialog box in which you can type the name of the node you want to find.You can search for a node in the Model, the Item Master, or in both locations.

Figure 9–1 The Find Node Dialog Box

The Find dialog box supports the use of **wild-card characters**. The wild-card for a complete string is % (percent sign). The wild card for a single character is _ (an underscore). If you need to search for a name that contains one of the wild-card characters, prefix the wild-card character with the escape character / (forward slash).

Table 9–6 provides examples of search results using wild-card characters.

Table 9–6 Searching for Model Nodes Using Wildcard Characters

Search target	Finds
Feature/_10%	Feature_1040, Feature_1041, Feature_10 abc
Feature/_10_	Feature/_104

9.6.3.1 Model Window Keyboard Shortcuts

Table 9–7 lists the keyboard shortcuts available in the Model window.

Table 9–7 Model Window Keyboard Shortcuts

Command	Shortcut
Cut	ctrl-x
Copy	ctrl-c
Paste	ctrl-v
Find	ctrl-f

These commands are also available from the **Edit** menu in the Model window.

9.7 Create Menu

9.7.1 Create Menu for the Model Module

Use this menu to create new nodes in your Model. Whether you can create a particular type of node depends on the type of node that is currently selected.

Table 9–8 lists commands on the Create menu.

Table 9–8 Model Window Create Menu Commands

Command	Description
New Product	Creates a new Product node under the currently selected node.
New Component	Creates a new Component node under the currently selected node.
New Feature	Creates a new Feature node under the currently selected node.
New Option	Creates a new Option node under the currently selected node.
New Total	Creates a new Total node under the currently selected node.
New Resource	Creates a new Resource node under the currently selected node.
New Model Reference...	Creates a new Model Reference node under the currently selected node. Opens the Models dialog to enable selection of a Model node defined in the Repository.

9.7.2 Create Menu for the Item Master

Table 9–9 lists the commands that appear on the Create menu when you are working in the Item Master. Use these commands to create new nodes in your Item Master.

Table 9–9 Create Menu for the Item Master

Command	Description
New Item	Creates a new Item node under the currently selected node.
New Item Type	Creates a new Item Type node under the currently selected node.

9.7.3 Create Menu for the Configuration Rules Module

Use this menu to create new configuration rules. Your rules are organized as nodes in the Configuration Rules tree. When you create a rule, Oracle Configurator Developer places a node for the rule in the folder you have selected. You can create as many rule folders as you need to organize them in a meaningful manner. Rule folders that you create can contain multiple types of rules.

Table 9–10 lists the commands that appear on the Create menu when you are working in the Configuration Rules module.

Table 9–10 Create Menu for the Configuration Rules Module

Command	Description
New Rule Folder	Creates a new rule folder.
New Logic Rule	Creates a new Logic Rule node.
New Numeric Rule	Creates a new Numeric Rule node.
New Comparison Rule	Creates a new Comparison Rule node.
New Property-based Compatibility	Creates a new Property-based Compatibility node.
New Explicit Compatibility	Creates a new Explicit Compatibility node.
New Functional Companion	Creates a new Functional Companion node.
New Design Chart	Creates a new Design Chart node.
New Rule Sequence	Creates a new Rule Sequence node

9.7.4 Create Menu for the User Interface Module

Use this menu to create new elements in your User Interfaces. Your elements are organized as nodes in the User Interface tree. Table 9–11 lists the commands that appear on the Create menu when you are working in the User Interface module.

Table 9–11 Create Menu for the User Interface Module

Command	Description
New User Interface	Creates a new User Interface in the User Interface Context Tree View, corresponding to the Product or Component selected in the Model tree.
New Text	Creates a new Text label under the selected screen node.
New Button	Creates a new Button under the selected screen node.
New Picture	Creates a new Picture under the selected screen node.
New Value Display	Creates a new Tagged Value Display under the selected screen node.
New Connections Listbox	Creates a new Connections Listbox under the selected screen node.

9.8 View Menu

The **View** menu is divided into upper and lower sections. The lower section selects from among the Configurator Developer modules, and corresponds to the buttons on the Module toolbar (see Section 9.8.1, "Module Toolbar" on page 9-12). The upper section presents choices that depend on what module is currently active.

Table 9–12 lists the commands that appear on the **View** menu when you are working in the User Interface module.

Table 9–12 View Menu for the User Interface Module

Command	Description
Preview	Opens the UI Editor for the selected UI screen node.
Model	Switches to the Model module.
Model > By Name / By Description	Changes the display of the nodes in the Model, grouping them by Type or by Description.
Configuration Rules	Switches to the Configuration Rules module.
User Interface Builder	Switches to the User Interface module.

Table 9–12 (Cont.) View Menu for the User Interface Module

Command	Description
Test / Debug	Launches the Oracle Configurator test environment you selected for the current Model. Choose **Tools > Options** to select a test environment.

Table 9–13 lists the commands that appear on the **View** menu when you are working in the Model module.

Table 9–13 View Menu for the Model Module

Command	Description
Model > By Name, By Description	View the Model structure either by each node's name or the description provided (if any).
Item Master > By Name, By Description, By Item Type and Name, By Item Type and Description	View Items in the Item Master by each Item's name, the description provided (if any), the Item Type and name, or the Item Type and description.

9.8.1 Module Toolbar

You can use the Module toolbar instead of the **View** menu to switch from one module to another.

Toolbar Icon	Command
	Go to the Model module.
	Go to the Configuration Rules module.
	Go to the User Interface module.

Toolbar Icon	Command
	Go to the Test module.

9.9 Tools Menu

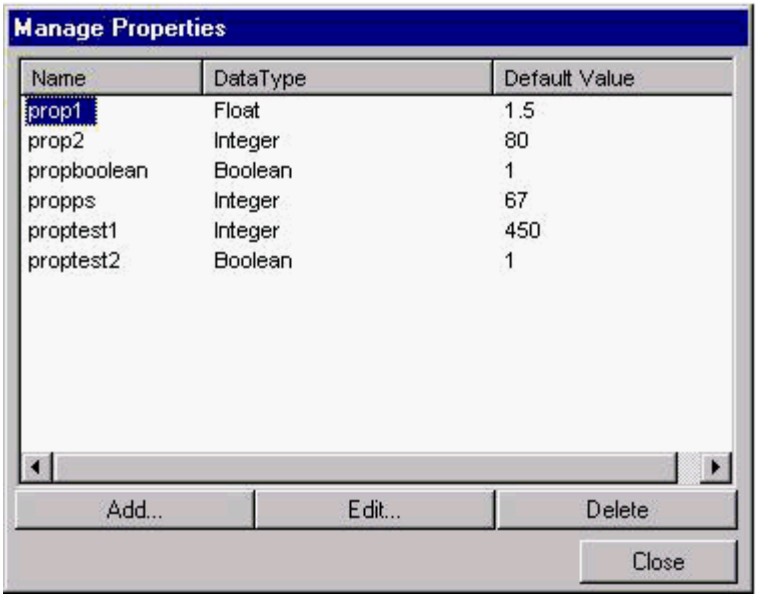
Use this menu to change a Model window session option, manage properties, or to automatically build elements of your application. Table 9–14 lists the commands that appear on the **Tools** menu when you are working in the User Interface module.

Table 9–14 Tools Menu for the User Interface Module

Command	Description
Generate Active Model	Generates a generic Active Model for the currently selected Model. This includes generating configuration rules.
RePopulate	Runs all Populators, refreshing your Model from the Item Master after it has been loaded with updated data from the import tables.
Manage Properties	Enables you to create, edit, or delete Properties. Use the Property Manager to delete a Property from your Model.
Options...	Opens the Options dialog for the current Oracle Configurator Developer session.

9.9.1 Manage Properties

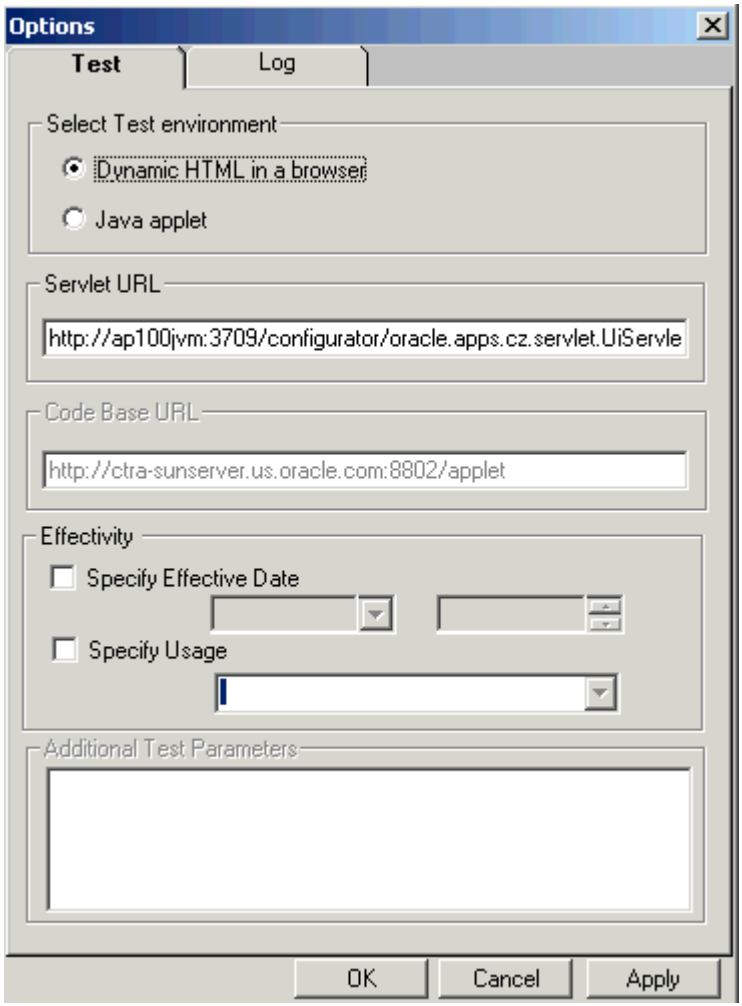
The Manage Properties dialog box enables you to add, edit, and delete Properties in your Model or Item Master. For more information about Properties, see Section 3.2, "Properties" on page 3-2.



9.9.2 Options

The **Options** dialog enables you to control display of the Configurator Developer Log Messages window, and select an environment for testing and debugging. To view this dialog, choose **Tools > Options** from the Model window.

Figure 9–2 The Options Dialog Box



9.9.2.1 Selecting a Test Environment

The **Test** tab of the Options dialog enables you to set up the environment in which you test your Model. Select one of the following environments:

- **Dynamic HTML in a browser**: The Configurator Developer Test module displays the Model and User Interface in your default Web browser.

- **Java Applet**: The Configurator Developer Test module displays the Model and User Interface as a Java applet in your default Web browser.

For both Dynamic HTML in a browser and Java Applet, you must supply the Oracle Configurator **Servlet URL**. The Servlet URL is the location where the servlet resides and is set up by the installer of the servlet. Oracle Configurator Servlet installation is described in detail in the *Oracle Configurator Installation Guide.*

The Servlet URL has the following syntax:

```
http://host:port/configurator/oracle.apps.cz.servlet.UIServlet
```

For example:

```
http://www.mysite.com:60/configurator/oracle.apps.cz.servlet.UiServlet
```

You can test that the specified servlet is running by entering a command with the following structure in the Location field of your browser:

```
URL of the Servlet?test=version
```

For example:

```
http://www.mysite.com:60/configurator/oracle.apps.cz.servlet.UiServlet?test=vers
ion
```

> **Note:** The property cz.uiservlet.versionfuncsavail determines whether you can test the response of the OC Servlet using this method. For more information about this property, see the *Oracle Configurator Installation Guide.*

If you select Java Applet as your test environment, you must also supply the **Code Base URL**. This is the URL from which the required class or Java archive files (.jar files) can be loaded. The URL has the form:

```
http://host:port/applet/
```

For example:

```
http://www.mysite.com:60/applet/
```

For more information about the Java archive files required to run the Java applet from Configurator Developer, see the *Oracle Configurator Installation Guide.*

Spx.ini File

The values you specify for the UI test environment, Servlet URL, and Code Base URL are stored in the spx.ini file. In this file, the value of the **Launch** parameter is `1` for Dynamic HTML in a browser and 2 for the Java Applet. The value of the **InitServletURL** parameter is the Servlet URL. The value of **InitCodeBaseURL** is the Code Base URL.

Although it is possible to specify the InitServletURL and InitCodeBaseUrl parameters by editing the spx.ini file, it is recommended that you enter this information in the Options window, as described in Section 9.9.2.1, "Selecting a Test Environment" on page 9-15.

If you are using Dynamic HTML in a browser as your test environment, you must also specify the **JdbcUrl** parameter in the spx.ini file. See the *Oracle Configurator Implementation Guide* for more information.

Any browser running the Oracle Configurator DHTML window must be set to enable JavaScript style sheets and must enable cookies. For more information, refer to your browser's online help.

9.9.2.2 Applying Effectivity

The Effectivity option enables you to view the Model based on the effective dates, Effectivity Sets, and Usages assigned to each node.

To limit the availability of nodes based on the date, enter an effective date and time. You can also control which nodes appear in the UI by specifying a Usage. If you do not want to hide nodes in the runtime UI based on a Usage, leave this option blank. See Section 3.3, "Effectivity" on page 3-4.

9.9.2.3 Additional Test Parameters

This section is reserved for future functionality.

9.9.2.4 Displaying the Log Messages Window

Open the **Log** tabbed region of the Options window to view the location of the log file to which messages are written, if one is currently in use. Check the **Show Datastore Log** box to display the Log Messages window. For more information on the Log Messages window, see Section 9.24, "The Log Messages Window" on page 9-55.

9.10 Help Menu

Use this menu to get information about Oracle Configurator Developer and how to use it to construct your configurator. Table 9–15 lists the options on the **Help** menu.

Table 9–15 Help Menu Options

Command	Description
Help Topics	Opens the Oracle Configurator Developer online help.
About Oracle Configurator Developer	Displays information about this version of Oracle Configurator Developer.

9.11 Tree Views

Oracle Configurator Developer displays the Model modules as tree structures called **tree views**. Tree views show how elements of the structure are related to each other, and which structure contain others. Branches of the tree, can be collapsed and hidden or expanded and displayed by using the (-) and (+) controls that appear on nodes that contain other nodes.

Within a tree, each element of the structure is a node. The top-level node is the *root*. When all of the tree has been collapsed or hidden, only the root node remains visible. Nodes in Oracle Configurator Developer tree views have names and graphical icons associated with them. The graphical icons provide visual cues that indicate the kind of information the node represents.

9.12 Node Attributes Present in all Tree Views

The Name and Description attributes are available for every node in all tree views.

9.12.1 Name

The **Name** field provides a name for the selected node. Oracle Configurator Developer assigns a default name based on the node's internal ID in the CZ schema. Enter a more informative name to help you identify the node. You can use the same name for different nodes, since the CZ schema's internal ID for each node is unique. This name and ID appear in the Model Report (see Section 9.24.1, "File Menu" on page 9-55).

9.12.1.1 Model Name

If you create a node with a Populator, then the name is set automatically to the name of the corresponding Item in the Item Master. See Section 4.4, "Using Populators" on page 4-8.

9.12.1.2 Item Master Name

When you create Items and Item Types in the Item Master by importing data, Oracle Configurator Developer assigns names based on the import tables.

When you create Items in the Item Master manually, Oracle Configurator Developer assigns a default name based on the node's internal ID in the CZ schema. You can enter a more descriptive name to identify Items more easily.

9.12.2 Description

Use the **Description** field to enter a description of the selected node. By default, this field is blank. A node's description does not appear in the runtime UI or the Model Report.

9.13 Tree Views for the Model Module

There are two tree views available for this module, the Model tree view and the Item Master tree view. The Attributes available in the Attributes View depend on which tree view is selected. Table 9–16 maps each tree view and its corresponding Attributes View.

Table 9–16 Tree Views and Attribute Display

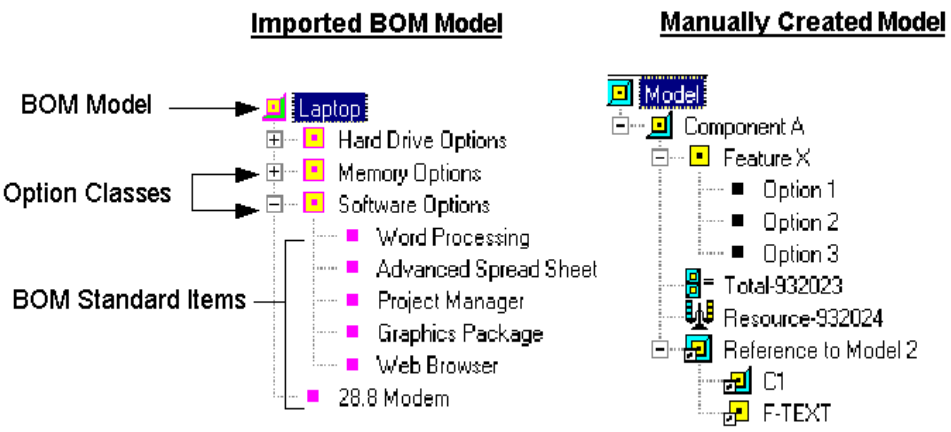
Tree View Selected	Attribute View displayed
Model	Model View on page 9-21
Item Master	Item Master Attributes on page 9-32

9.13.1 Model Tree View

You use the Model Tree to create and modify the Model nodes and create References. The Model tree view consists of nodes that represent the related Products, Components, Features, Options, Resources, and Totals. Nodes that represent Model references and their children are differentiated by a small arrow that appears as part of the icon.

Figure 9–3 shows the difference between an imported BOM Model and structure that you create in Configurator Developer.

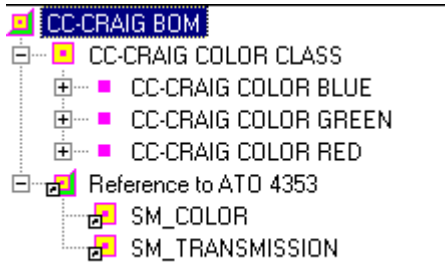
Figure 9–3 Imported BOM and Manually Created Model Structure



Note: You can view the nodes in the Model either by name or by description using options on the View menu.

The Model tree view uses a different color to distinguish the nodes of an imported BOM Model from nodes that you create in Configurator Developer. Nodes that represent BOM Model References and their child nodes are differentiated by a small arrow that appears as part of the icon, as shown in Figure 9–4.

Figure 9–4 BOM Model and Reference Nodes



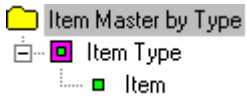
9.13.2 Item Master Tree View

The Item Master tree view consists of nodes that represent the Items and Item Types in the Item Master. Using the **View** menu, you can view Items in the Item Master by name, description, or by Item Type.

For more information about Items and Item Types, see Section 3.4.1, "Item Types and Imported BOM Properties" on page 3-8.

Figure 9–5 shows the icons associated with each of the nodes in the Item Master tree view.

Figure 9–5 Item Master Nodes



9.14 Model View

The Model view appears in the upper left pane of the Model window, regardless of which Configurator Developer module is selected. It provides different sets of Attributes for the selected node, depending on the module in which you are working and the type of node that is selected.

9.14.1 Attributes of Nodes Created in Configurator Developer

Table 9–17 shows the information that is available in the Attributes View when you are working in the Model module and select a node created in Configurator Developer (that is, a non-BOM node).

Table 9–17 Model Module Attributes for Non-BOM Nodes

Attribute	Node Type					
	Models	Components	Features	Options	Totals and Resources	Connectors
Name on page 9-18	X	X	X	X	X	X
Description on page 9-19	X	X	X	X	X	X
Definition on page 9-28						X
Instances on page 9-29	X	X				
Type on page 9-23			X			
Visibility on page 9-28	X	X	X	X	X	X
Properties on page 9-30	X	X		X		
Populators on page 9-31	X	X	X			
Effectivity on page 9-31	X	X	X	X	X	X

9.14.2 Attributes of Imported BOM Nodes

The following attributes are available in the Attributes View when you are working in the Model module and select an imported BOM Model, BOM Option Class, or BOM Standard Item:

- "Name" on page 9-18
- "Description" on page 9-19
- "Definition" on page 9-28
- "Instances" on page 9-29 (appears only if node is an instantiable BOM Model)
- "Properties" on page 9-30
- "Visibility" on page 9-28
- "Effectivity" on page 9-31

Note that when you are viewing an imported BOM Model in the Model view, BOM Option Class nodes are equivalent to Feature nodes, and BOM Standard Items are equivalent to Feature Options. See Section 3.4.3, "Viewing Imported BOM Model Nodes" on page 3-10.

9.14.3 Type

This attribute specifies the type of data represented by the selected Feature node. Choose one of the selections in the **Data Type** list. To view examples of how each type of Feature appears in the runtime UI, see Section 6.1.3.1, "Data Fields" on page 6-5.

Table 9–18 lists the available Feature Data Types and a description of each.

Table 9–18 Feature Data Types

Data Type	Description
List of Options on page 9-24	A List of Options node, displayed as children of the selected node. You can create Options automatically by defining a Populator or create them manually using the Create menu. See Using Populators on page 4-8 or Create Menu on page 9-9.
Integer Number on page 9-25	A whole number, such as 3 or 127. Also known as a Numeric Feature.

Table 9–18 (Cont.) Feature Data Types

Data Type	Description
Decimal Number on page 9-26	A number with a decimal part, such as 3.14159. Sometimes referred to as a floating point number. Also known as a Numeric Feature.
True / False on page 9-26	Has two values, True and False. Sometimes referred to as a Boolean Feature.
Text on page 9-26	A string of characters.

Warning: Use caution when changing the **Data Type** of a List of Options Feature. If you change the **Data Type**, **Configurator Developer** deletes any **Option**s or **Populator**s associated with the **Feature**. Additionally, any configuration rules using the deleted **Option**s become invalid. (An error message indicates that the rule is invalid when you view its definition or generate the Active Model.) **Configurator Developer** displays a warning message when you attempt to modify the **Data Type** and describes the consequences of the change.

Each Feature requires additional information, depending on its **Data Type**. This information is described below.

List of Options

- **Number of Selections:** Indicates the **Minimum** and **Maximum** number of Feature Options that the end user can select. For example, if the Feature contains five Options and you set the **Maximum** to 3, only three of the five Options can be selected at runtime. Additionally:
  - The default value for both the **Minimum** and **Maximum** is 1.
  - A **Minimum** of 1 means that at least one Option must be selected. In other words, the Feature is required and must be selected at runtime. (If the end user does not select at least one of the Feature's Options, the Feature will be unsatisfied and the configuration will be incomplete).
  - A **Maximum** of 1 means the end user can select only one Option, which means that selecting one Option of a Feature prevents then end user from selecting any other Options of that Feature.

- A **Minimum** of 0 means selecting an Option from this Feature is optional. If you attempt to set the **Maximum** to be less than the **Minimum**, the **Minimum** and **Maximum** are set to the same value.
- You can indicate that a Feature has no maximum value by deleting the numeric value from the **Maximum** field.

- **Counted Options:** Select the **Counted Options** box if you want end users to be able to specify a quantity for each Feature Option. For example, if the Options are types of candy bars and the **Counted Options** box is selected, the end user can specify six milk chocolate bars and six dark chocolate bars. You can also use the number of Options entered (the Option count) in a Numeric rule. See Section 5.7, "Numeric Rules" on page 5-21.

  If the Feature has a Maximum of 1, or if the user must not specify more than one of each Option, do not select the **Counted Options** box.

> **Note:** The **Minimum** and **Maximum** number of selections does not constrain the value an end user can enter when **Counted Options** is selected.

Integer Number

- **Range**: The **minimum** and **maximum** values allowed for this Feature at runtime. An Integer Feature that has a minimum value greater than or equal to 0 is treated at runtime as a Count Feature. Count Features are not available in Configurator Developer; they are created at runtime and are represented in the CIO. For more information, see the *Oracle Configuration Interface Object (CIO) Developer's Guide*.

  A Count Feature has a logic state in addition to its numeric value, and if it is set to 0 in a runtime Oracle Configurator, its logic state becomes Unknown. Count Features can participate in Logic Rules, while Integer and Decimal Features cannot. Count Features behave like Counted Options in an Option Feature (see **Counted Options**, above, for more information).

- **Initial Value**: The value that this Feature has when the runtime Oracle Configurator starts. This value must be between the **minimum** and **maximum** values, inclusive. See Section 9.14.3.1, "Initial Value" on page 9-26.

A Features with this **Data Type** is also known as a Numeric Feature.

Decimal Number

- **Range**: The **minimum** and maximum values allowed for this Feature at runtime, in decimal format (for example, 5.0 to 13.5). Numeric Features display up to 9 values after the decimal point (for example, .123456789).
- **Initial Value**: The value that this Feature has when the runtime Oracle Configurator starts. This value must be between the **minimum** and **maximum** values, inclusive. See Section 9.14.3.1, "Initial Value" on page 9-26.

A Features with this **Data Type** is also known as a Numeric Feature.

True/False

- **Default Value**: The value that this Feature has when the configuration session starts. This is also the value the Feature reverts to when it is changed during a configuration session, but is not set to a specific value. For example, a configuration rule states that A implies B. If B has a default value of None, it becomes Logic True when the end user selects A and returns to a logic state of Unknown if A is later deselected. If B has a default value of Logic True, it retains this logic state when A is selected and even if A is later deselected. A True / False Feature is also known as a Boolean Feature.

  A True / False Feature (Boolean) with a **Default Value** of **False** appears blank at runtime (that is, as if its logic state were Unknown) when the configuration session begins. However, any configuration rules you have defined respect the default setting you specified and consider the Feature to be Logic False. A True / False Feature with a **Default Value** of **True** is selected (that is, it has a logic state of Logic True) when the configuration session begins.

  A value of None is recommended because choosing either True or False can adversely effect runtime performance. See Section 9.14.3.1, "Initial Value" on page 9-26.

Text

- **Text Value:** The value that this text Feature has when the runtime Oracle Configurator starts.

9.14.3.1 Initial Value

Use this field to set a value that the selected node has before any quantities are contributed or consumed. By default, the **Initial Value** for Totals and Resources is blank (null) in Configurator Developer. If you leave this field blank, the initial value displays as a 0 (zero) at runtime.

When the initial value of a Total is zero, configuration rules that involve the Total do not propagate.

For Boolean Features (**Data Type** is **True/False**), the initial value is essentially a default, which like all defaults can be overridden by the end user. Therefore, the end user can select a Boolean Feature that is initially Logic False and it will appear as User True in the runtime UI. For more information, see "True/False" on page 9-26.

Oracle Configurator Developer uses the following procedure to set and update initial values (for example, when an end user makes a selection in the runtime User Interface):

1. The initial values specified in Configurator Developer are set in the UI.
2. When an end user makes a selection in the Configurator window, the system retracts the initial values and sets them to "". It then changes the selected option to User True.
3. The system applies any rules in which the selection is a participant.
4. The system changes the logic state of any options that are also participants in the rule and are therefore affected by the end user's selection.

There may be situations in which the runtime UI selects an option but the criteria for making the selection is not readily apparent. For example, you define three Boolean Features, F1, F2, and F3 and set the **Initial Value** for both F2 and F3 to False. You then define a Logic rule with an Implies relation that states "F1 Implies Any True (F2, F3). At runtime, the initial value for all of these Features is false. When the end user selects F1, the values for all three Features is retracted, set to Unknown, and then F1 is set to true. The system then applies the rule and sets either F2 or F3 to false and the other to true. (This is because the Implies rule states that if the end user selects F1, then either F2 or F3 must also be selected.)

If the system selects F2, the user can override it by selecting F3, but there is no setting in Configurator Developer to specify which option should be selected (set to true) and which should be set to false when the rule is initially triggered.

Using the Features and Logic rule described in the previous example, consider the case in which the **Initial Value** is set to None for Features F1, F2, and F3. At runtime, all of the Features are set to Unknown. When the end user selects F1, F2 and F3 are still Unknown; this is because the Implies relation uses Any True and at this point the system does not know which Feature should be selected. If the end user selects F2, the rule sets F3 to Logic False (the reverse is also true).

Another situation that may cause confusion is when a Feature has a minimum quantity that is greater than the **Initial Value** that you specify. For example, you create a Numeric Feature and enter a **Minimum Number of Selections** of 3 and an **Initial Value** of 0. When you generate the runtime UI, the **Minimum Number of Selections** overrides the **Initial Value** and the Feature is set to 3 when the configuration session begins.

9.14.4 Definition

The Definition attribute provides a basic definition of the selected imported BOM node. Information in this section is read-only, since these attributes are defined and can only be modified in Oracle Bills of Material.

This attribute displays the **BOM Item Type**, **Minimum Quantity**, **Maximum Quantity**, **Default Quantity** for the selected BOM item. The **BOM Item Type** field indicates whether the selected item is an BOM Model, a BOM Option Class, or a BOM Standard Item. For details about the **Minimum**, **Maximum**, and **Default Quantity** values, see Section 3.4.5, "Quantity Cascade Calculations" on page 3-12.

The **Definition** attribute also contains check boxes that indicate whether:

- Optional children of the selected BOM item are mutually exclusive (see Section 3.4.2 on page 3-9)
- The selected BOM item is required in the configuration when its parent is selected (Section 3.4.2 on page 3-9)
- An Oracle Configurator end user can enter a decimal quantity for this BOM item (see Section 3.4.3.3 on page 3-11)
- The selected BOM item is **Trackable**. If this check box is selected, information about the item is recorded in Oracle Install Base. This setting is relevant only to Models that allow updates to installed configurations. For more information, refer to Telecommunications Services Ordering documentation.

The **Definition** attribute also appears if the selected node is a Connector. In this case, the **Connection Required** box indicates whether an end user must connect the selected node and its target at runtime to create a valid configuration. For more information, see Section 4.3.5 on page 4-6.

9.14.5 Visibility

By default, all nodes appear in the User Interface. However, you can control whether a non-BOM node and any of its children appear in the generated User

Interface using the **Visibility** attribute in the Model module. Deselect the **Display in User Interface** box to prevent the selected node from appearing at runtime.

The Visibility attribute is read-only for imported BOM nodes, but some imported nodes might not appear at runtime because of their effective dates. For more information, see Section 6.3.5, "Hiding Objects in the Runtime User Interface" on page 6-38.

You can override the default Visibility setting when you generate a new UI by selecting **Show all nodes** in the Generate UI window. See Section 6.2, "Generating a New User Interface" on page 6-16.

9.14.6 Instances

Controls the **Minimum** and **Maximum** number of **Instances** of a Model, Component, or BOM Model that can be present in a valid configuration. An Oracle Configurator user can add or delete instances at runtime by clicking an **Add** or **Delete** button. See Section 6.1.3.3, "Add and Delete Buttons" on page 6-8.

To make a selection mandatory in your application, set the **Minimum** to 1 or more. To make a selection optional, set the **Minimum** to 0 (zero). The default setting is a minimum and maximum of 1, which allows the end user to configure only one instance of the selected Model, Component, or BOM Model at runtime.

For example, if a Component's **Minimum** is 1 and its **Maximum** is 10, at least 1 instance of the Component must exist in the configuration, but there may be as many as 10 instances total.

It is possible to define configuration rules that modify the minimum and maximum number of instances allowed at runtime. See Section 5.7.5, "Building Rules Using Node Properties" on page 5-26.

For more information about instantiating components at runtime and the effect that changing the Instances values has on a saved configuration, see Section 3.5, "Multiple Instantiation in Solution Models" on page 3-14.

Note: After modifying the **Minimum** or **Maximum**, you must refresh the User Interface so the new setting is used in the runtime UI.

9.14.7 Properties

Use this region to add or modify attributes of the selected node. Properties provide additional information about a node, such as weight, diameter, or voltage.

For a general description of Properties, see Section 3.2, "Properties" on page 3-2.

For information about imported BOM Properties, see Section 3.4.1, "Item Types and Imported BOM Properties" on page 3-8.

9.14.7.1 Adding and Modifying Properties

You can modify and delete only Properties that you assign to nodes in Configurator Developer. Properties imported with a BOM Model, BOM Option Class, or BOM Standard Item are read-only.

To add a Property:

1. Select a node in the Model view.
2. Expand the Properties attribute, then click Add.
3. In the Add Property dialog, select an existing Property, or click New Property to create one. If you create a new Property, enter a name and a default value.

To modify a Property:

1. Select a node in the Model view, then expand the **Properties** attribute.
2. Select a Property from the list, and then click **Edit**.
3. In the Edit Property dialog, change the Property value, or click **Default** to restore the Property's default value.

> **Note:** When you modify a Property value using this method, the change affects only the selected node. If you want the change to apply to all nodes assigned to a specific Property, use the Manage Properties dialog. See Section 9.9.1, "Manage Properties" on page 9-13.

To delete a Property:

1. Select a non-BOM node in the Model view, then expand the **Properties** attribute.
2. Select a Property from the list, then click **Delete**.

9.14.8 Populators

This attribute lists the Populators associated with the selected node and allows you to add, edit, or delete them. To add a Populator, click **Add**. To modify the definition of a selected Populator, click **Edit**. To delete a selected Populator, click **Delete**.

For general information about Populators, see Section 4.4 on page 4-8.

9.14.9 Violation Message Attribute

Use this attribute to define the violation message that appears when the selected Resource is over-consumed. The default message is "The resource *resource-name* is over-consumed." See Section 4.3.4, "Creating a Total or Resource" on page 4-5.

9.14.10 Effectivity

The **Effectivity** attribute enables you to control the availability of a Model node or configuration rule by assigning a range of effective dates, a Usage, or both. For more information, see Section 3.3, "Effectivity" on page 3-4.

9.14.10.1 Effectivity Sets

To assign an Effectivity Set to a node:

1. Select the node.
2. Select the **Member of Effectivity Set** box.
3. Click the dialog button **(...)** to open the **Choose Effectivity Set** dialog. This dialog lists all Effectivity Sets in the current CZ schema.
4. Select an Effectivity Set from the list and then click **OK**.

   To create a new Effectivity Set, click **New** then enter a name, description, and effective date range. (All Effectivity Set names must be unique.) When you create an Effectivity Set using this method, Configurator Developer creates a new Effectivity Set node at the top level in the Repository window (not in a Folder). If you want to move the Effectivity Set into a Folder, you must do so in the Repository window.

   To edit an existing Effectivity Set, you must open it from the Repository window. See Section 8.5, "File Menu" on page 8-6.
5. Click **OK**.

You can also create Effectivity Sets in the Repository window. See Section 8.9, "Create Menu" on page 8-8.

9.14.10.2 Dates

Click the Dates dialog button **(...)** to assign a range of effective dates to the selected node. This option is disabled if the **Member of Effectivity Set box is selected**. Select one of the following date options:

- Always Effective
- Never Effective
- Effective From / To - Enter start and end date and time, or **No Start Date,** or **No End Date**.

Although you can specify a very wide range of dates when entering a start and end date, this range is limited. For more information, see the *Oracle Configurator Implementation Guide*.

9.14.10.3 Usages

Click the Usages dialog button **(...)** to open the **Model Usages** dialog. This dialog lists all of the Usages you have defined. In this window you can:

- Click **Uncheck All** to deselect all Usages, then choose one or more Usages to assign to the selected Model node. By default, Configurator Developer assigns any Model node or rule that you create to all Usages.
- Click **Check All** to select all Usages.
- Click **Edit** to modify the selected Usage.
- Click **Add** to create a new Usage. When you create a Usage using this method, Developer creates a new Usage entity at the top level in the Repository window (not in a Folder). If you want to move the Usage into a Folder, you must do so in the Repository window. You can create a maximum of 64 Usages. (You can also create Usages in the Repository window. See Section 8.9, "Create Menu" on page 8-8.)

For more information, see Section 3.3.2, "Usages" on page 3-6.

9.15 Item Master Attributes

The Item Master is a set of your enterprise data, structured into Items and Item Types, that is the primary source of data for your application.

Table 9–19 shows the attributes that are available in the Attributes View when you are working in the Model module and select a node in the Item Master tree. Note that different attributes are available for different types of nodes.

Table 9–19 Item Master Attributes

Attribute	Node Type	
	Item Types	Items
Name on page 9-18	X	X
Description on page 9-19	X	X
Type / Properties on page 9-33		X
Properties on page 9-30	X	

9.15.1 Type/Properties

Controls the Properties of Item and Item Type nodes, and the Item Type of Item nodes.

When you create Items in the Item Master by importing data, Oracle Configurator Developer creates Item Types based on the import tables.

To change the Item Type of the selected Item, click **Change** and choose an Item Type from the Item Types dialog. When you close the dialog, the Item moves to the chosen Item Type. This is the only way to change the Item Type of an Item.

For information about changing the Properties of Item and Item Type nodes, see Section 9.14.7, "Properties" on page 9-30.

9.15.2 Properties

See Section 9.14.7, "Properties" on page 9-30.

9.16 Tree Views for the Configuration Rules Module

There are two tree views available for this module: the Model tree view and the Configuration Rules tree view. The attributes available in the Attributes View depend on which Tree view is selected. Table 9–20 maps each Tree View and its corresponding Attributes View.

Table 9–20 Tree Views and Attributes in the Configuration Rules Module

Tree View Selected	Attributes View Displayed
Model	Model Attributes for the Configuration Rules Module on page 9-35

Table 9–20 (Cont.) Tree Views and Attributes in the Configuration Rules Module

Tree View Selected	Attributes View Displayed
Configuration Rules	Configuration Rules Attributes on page 9-36

9.16.1 Model Tree View

You drag-and-drop nodes from the Model tree to the configuration rules that you are creating.

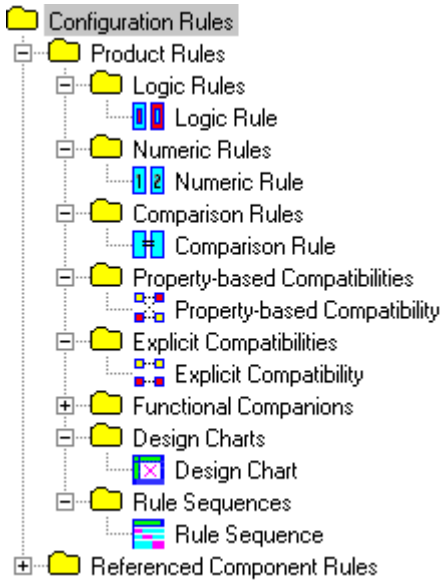
9.16.2 Configuration Rules Tree View

The Configuration Rules tree view consists of nodes that represent the rules, rule types, and rule folders associated with a Model. Create **rule folders** with descriptive names to better manage your rules more easily. Rule folders that you create can contain more than one type of rule. You can use standard drag-and-drop or cut, copy, and paste operations to move or copy a rule from one folder to another.

Use the **View** menu to view your rules by name, type, or by folder.

Figure 9–6 shows the icons associated with each type of configuration rule.

Figure 9–6 Configuration Rule Icons



9.17 Model Attributes for the Configuration Rules Module

The Model view is available in the left upper pane of every Oracle Configurator Developer module. It provides different sets of attributes for the selected node of your Model, depending on which module you are working in.

Table 9–21 shows the attributes that are available in the Attributes View when you are working in the Configuration Rules module and select a node in the Model tree. Note that different attributes are available for different types of nodes.

Table 9–21 Model Attributes in the Configuration Rules Module

Attribute	Node Type					
	Models	Components	Features	Options	Totals	Resources
Name on page 9-18	X	X	X	X	X	X
Description on page 9-19	X	X	X	X	X	X

Table 9–21 (Cont.) Model Attributes in the Configuration Rules Module

Attribute	Node Type					
	Models	Components	Features	Options	Totals	Resources
Instances on page 9-29	X	X				
Type on page 9-23			X			
Initial Value on page 9-26					X	X
Visibility on page 9-28						
Properties on page 9-30			X			
Associated Rules on page 9-36	X	X	X	X	X	X
Effectivity on page 9-31	X	X	X	X	X	X

9.17.1 Associated Rules

Lists the configuration rules in which the selected node is a participant. To edit an associated rule, select it from the list, and then click **Go To**. When you do this, the rule displays in the Context Tree View and the rule definition appears in the Attributes View.

To edit or delete an associated rule:

1. Select it from the **Associated Rules** list.
2. Click **Go To**. The rule displays and is selected in the Context Tree View.
3. Right click and select the option you want (for example, **Cut**, **Copy**, **Rename**, **Delete**, or **Add to Sequence**.

9.18 Configuration Rules Attributes

Table 9–22 shows the attributes that are available in the Attributes View when you are working in the Configuration Rules module and select a node in the

Configuration Rules tree. Note that different attributes are available depending on the type of node or rule type you select in the Model structure.

Table 9–22 Configuration Rules Attributes

Attribute	Node Type						
	Logic Rule	Numeric Rule	Comparison Rule	Property-Based Compatibility	Explicit Compatibility	Functional Companion	Design Chart
Name on page 9-18	X	X	X	X	X	X	X
Description on page 9-19	X	X	X	X	X	X	X
Parameters on page 9-37				X	X		
Definition on page 9-38	X	X	X	X	X	X	X
Violation Message on page 9-38	X	X	X	X	X		X
Unsatisfied Message on page 9-38	X		X				
Effectivity on page 9-31	X	X	X	X	X		X

9.18.1 Parameters

The **Participants** list in the **Parameters** attribute lists the configurable elements of your Model that are used in a Property-based Compatibility or Explicit Compatibility rule. Participants must be List of Options Features, BOM Option Classes, or both.

To create a rule, drag an Option Feature or BOM Option Class node to the **Participants** list. Dragging and dropping with the **right** mouse button gives you the option of cancelling the operation.

Note: The **Effects** list is not currently implemented.

9.18.2 Definition

Use this attribute to define the selected rule. The exact steps for rule definition differ for each type of rule.

Whatever type of rule you are creating, if you position the cursor over a node name in the rule definition, the full path to that node appears as a **tooltip**. This information helps to locate Model nodes and avoid confusion, which can occur if you are working in a large Model (one that has many nodes).

See the following sections for information on building each type of rule:

- "Logic Rules" on page 5-21
- "Numeric Rules" on page 5-21
- "Comparison Rules" on page 5-29
- "Property-based Compatibilities" on page 5-32
- "Explicit Compatibilities" on page 5-35
- "Design Charts" on page 5-40

9.18.3 Violation Message

Controls the message displayed if the end user makes an invalid selection that violates the rule. Select one of the following:

- Select **Rule Name** display only the name of the rule.
- Select **Rule Description** to display the description you entered.
- Select **Custom** and enter the text you want to display at runtime when the rule is violated. Choose this option to provide specific, detailed information to help the end user create a valid configuration.

9.18.4 Unsatisfied Message

Use this attribute to specify the information to display when the selected Logic or Comparison rule is "unsatisfied" at runtime. By default, Configurator Developer does not display a message when a rule has this status. For more information, see Section 5.1.4, "Unsatisfied Rules" on page 5-5.

The options within this attribute are the same as those listed in Section 9.18.3, "Violation Message" (except the **None** option, which means do not display a message when the rule is unsatisfied).

9.19 Tree Views for the User Interface Module

There are two tree views available for this module, the Model tree view and the User Interface tree view. The attributes available in the Attributes View depend on which Tree view is selected. Table 9–23 maps each Tree View and its corresponding Attributes View.

Table 9–23 User Interface Module Tree Views

Tree view	Attributes View
Model	Model Attributes for the User Interface Module on page 9-40
User Interfaces	User Interface Attributes on page 9-41

9.19.1 Model Tree View

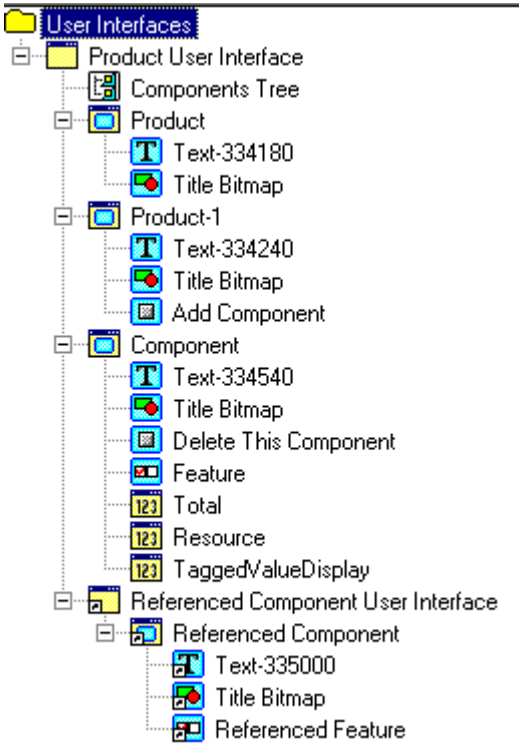
You use the Model tree to create and modify the nodes that your application's User Interfaces are based on.

9.19.2 User Interface Tree View

You use the User Interface tree to organize your the nodes of your User Interface definitions, which are based on the nodes in the Model.

The nodes in the User Interface tree view represent the User Interface, Components Tree, Screens, Pictures, Text, Buttons, Feature Controls, and Tagged Value Displays, and Referenced UIs for each UI. Figure 9–7 shows the icons associated with each of these nodes.

Figure 9–7 User Interface Tree View



9.20 Model Attributes for the User Interface Module

The Model view is available in every module of Oracle Configurator Developer, in the left upper pane. It provides different sets of attributes for the selected node of your Model, depending on which module you are working in.

Table 9–24 shows the attributes that are available in the Attributes View when you are in the User Interface module and select a node in the Model tree.

Table 9–24 User Interface Model Tree Attributes

Attribute	Node Type					
	Model	Component	Feature	Option	Total	Resource
Name on page 9-18	X	X	X	X	X	X

Table 9–24 (Cont.) User Interface Model Tree Attributes

Attribute	Node Type					
	Model	Component	Feature	Option	Total	Resource
Description on page 9-19	X	X	X	X	X	X
Associated UI Nodes on page 9-41	X	X	X	X	X	X

9.20.1 Associated UI Nodes

Displays a list of the User Interface nodes associated with the selected node. Click **Go To** to view the corresponding node in the User Interface Context Tree View.

9.21 User Interface Attributes

Table 9–25 shows the attributes that are available in the Attributes View when you are working in the User Interface module and select a node in the User Interface tree. Note that different attributes are available for different types of nodes.

Table 9–25 Attributes in the User Interface Tree

Attribute	UI Node Type		
	User Interface	Components Tree	Screen
Name on page 9-18	X	X	X
UI Model Object on page 9-42	X		X
Description on page 9-19	X	X	X
Version on page 9-43	X		
Defaults on page 9-43	X		
Pricing Display on page 9-44	X		
Styles on page 9-45			X
Definition on page 9-45		X	

Table 9–26 shows the attributes available for each UI node type.

Table 9–26 Attributes for Specific User Interface Node Types

Attribute	UI Screen Node Type						
	Text	Picture	Button	Feature Control	Tagged Value Display	Connector	Connections Listbox
Name on page 9-18	X	X	X	X	X		
UI Model Object on page 9-42				X			
Description on page 9-19	X	X	X	X	X	X	X
Definition on page 9-45		X	X	X	X	X	X
Label on page 9-49	X		X	X		X	X
Option Display on page 9-50				X			
Option Sorting on page 9-52				X			
Layout on page 9-52	X	X	X	X	X	X	X
Visibility on page 9-45				X			

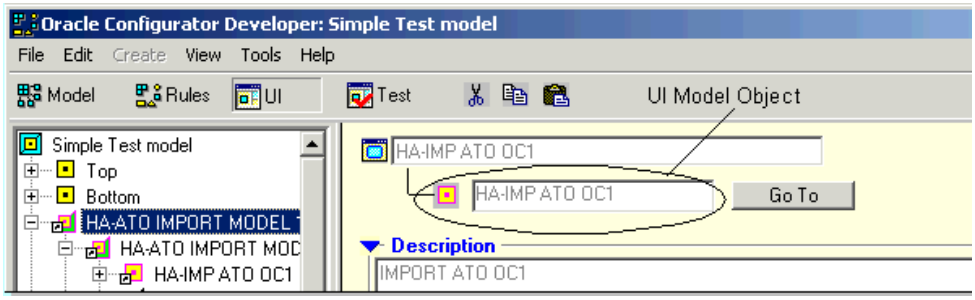
9.21.1 UI Model Object

This field appears next to the **Go To** button in the User Interface module, and displays a reference to the Feature, Component, or Model node that the currently selected UI node represents.

For example, the **Visibility** attribute for Component 1 is set so that it will appear in the runtime UI. Configurator Developer generates a UI Screen node called "Component 1" when you generate a new User Interface. When you go to the UI module and select this node, "Component 1" appears in the UI Model Object field. Clicking **Go To** displays and selects Component 1 in the Model view, and lists any other UI nodes that are associated with this Component.

This field is not labeled in Oracle Configurator Developer, as shown in Figure 9–8.

Figure 9–8 UI Model Object Field



9.21.2 Version

This attribute provides information the selected UI, such as creation date and UI style. Select the root node of the UI to view this information (for example, Sample Test Model 1 User Interface).

9.21.3 Defaults

This attribute specifies default values to be used throughout the UI, unless you override them with specific choices for particular screens. The UI attributes for which you can specify default values are:

- Look and feel
- Appearance of borders around BOM UI objects
- Text **font**, including style, size, and color
- Screen background **color** or **picture**
- Logic state display icons and text color
- Whether unselectable Feature Options and Feature Controls should be hidden (see Section 6.1.5, "Controlling Visibility" on page 6-10)
- Display order of Feature Options
- Display of node names in runtime messages
- Space allocation for the runtime Navigation tree

See Section 6.3.1, "Customizing the User Interface Default Settings" on page 6-21.

9.21.4 Pricing Display

The **Pricing Display** attribute displays settings at the root node of every customizable User Interface. Use the settings in this section to specify whether list prices or selling prices display at runtime, and how often list prices are updated.

9.21.4.1 Item Price Display

This setting determines which prices appear in the runtime UI. Choose one of the following:

- **List Prices**: Display the unit list prices for each item. This price does not change during a configuration session.
- **Selling Prices**: Display the unit selling price of all selected items. Items that are not yet selected display their unit list price. Selling prices can change during a configuration session, depending on the setting of the **Price Update** option (see below).
- **None**: Choose this option if you do not want to display either list prices or selling prices in the UI.

Note: The OC Servlet property `cz.activemodel` must be defined to display list prices in the either the DHTML or Java applet. For more information, see the *Oracle Configurator Installation Guide*.

9.21.4.2 Price Update

This setting indicates how often the item selling price is updated in the runtime UI. List prices do not change during a configuration session, but selling prices may change as items are added to the configuration.

Choose one of the following:

- **On Demand**: Update prices only upon the end user's request. This is the default value. If you choose this option, you must add a way for the user to update prices from the UI. For example, you can create a button and set its action to **Update Prices**. See Section 6.3.4.4, "Adding Buttons to a Screen" on page 6-28.

- **On Page Load**: Update prices whenever a new Component page is displayed or the end user requests an update.
- **Always**: Update prices each time the end user makes a selection, requests a pricing update, or displays a new component page.

9.21.5 Styles

This attribute controls **Background Color**, **Background Picture** and **Font** for the selected UI screen.

9.21.6 Definition

Which settings are available in the **Definition** Attribute varies depending on the selected User Interface node. The following list describes the possibilities.

9.21.6.1 Button Style

Use this attribute to specify the appearance of the selected button in the runtime UI. Choose **Rounded Right**, **Rounded Left**, **Rounded Both Sides**, or **Image**. For more information about these settings, see Section 6.3.4.4 on page 6-28.

9.21.6.2 ALT Text

Use this attribute to enter text that you want to appear when an end user positions the cursor over a UI object in the runtime UI. For example, if the selected UI node is a button with the Launch URL action, you might enter "Click here to view the XYZ Corp Web site" in the **ALT Text** field.

9.21.6.3 Font

This attribute specifies a text font. Click the dialog button (**...**) in the **Font** field to open the Font dialog, then select a font and its related attributes.

9.21.6.4 Picture

This attribute specifies the pathname to a graphics file. Click the dialog button (...) in the **Picture** field to open the standard Windows Open dialog. Select a GIF or JPEG image from %ORACLE_HOME%\Osp\Shared\ActiveMedia\ on the machine running Configurator Developer, then click **Open**. The full path of the file appears in the **Picture** field. To remove an image file, highlight the field contents and delete the path.

To be available to the runtime Configurator, the image file must also exist in the directory specified by OA_MEDIA for the OC Servlet. See the *Oracle Configurator Implementation Guide* for more information.

> **Note:** Oracle Configurator Developer supports only GIF (*.gif) and JPEG (*.jpg) files in the runtime DHTML UI. Bitmap files (*.bmp) are not supported.

9.21.6.5 Borders

This attribute specifies the type of border to use. Select **None** or **Fixed Single** from the dropdown list in the **Borders** field.

9.21.6.6 Background Color

This attribute specifies a color. Select the **Use Default** option box or click the dialog button (...) in the **Color** field to open the Color dialog. Select a basic color or click **Custom Color** to define a unique color, then click **OK**.

9.21.6.7 Action

This attribute specifies an action that takes place when the user clicks a button or image in the User Interface. Select one of the following options from the **Action** dropdown list. The default action is **None**.

- **None**: No action.
- **Add Component**: Add an instance of a Component or Model to the configuration. You can specify an Add Component action only to the parent UI screen of the Component or Model to add. For example, Model M1 contains the Component C1. To be able to add new instances of C1 at runtime, add a button to Model M1's UI screen and choose C1 from the **Reference** list.

  Select the Component or Model to add from the **Reference** selection list. This list contains Components and Models that:

  - Are children of the Model node corresponding to the selected UI screen
  - Can have variable numbers of instances, meaning that they have a minimum and maximum number of **Instances** is not equal (see Section 3.5, "Multiple Instantiation in Solution Models" on page 3-14)
- **Delete Component**: Delete an instance of a Component or Model from the configuration. You can add a Delete Component action only to the UI screen of the Model or Component to delete. The **Reference** selection list for a Delete

Component action contains only the Model node corresponding to the selected UI screen node.

> **Note:** Under certain conditions, Configurator Developer automatically creates an Add *Model Name* or a Delete *Model Name* button on each UI screen that represents a referenced Model. See Section 6.1.3.3, "Add and Delete Buttons" on page 6-8.

- **Go To Screen**: Navigate to a specific screen in the current UI. To select a screen, click the dialog **(...)** button next to the **Reference** field, select a screen from the Choose Target Screen window, then click **OK**. The Choose Target Screen window displays all screens in the selected UI.
- **Launch URL**: Open a specific Web document in a separate browser window. You may want to use this option to enable the end user to view additional information about an option before adding it to the configuration. The method you use to specify the URL depends on the type of UI node that is selected. For example:
  - If you are specifying an action for a Button, Picture, Text, or the UI caption of a Feature Control, Value Display, or BOM Standard Item, enter a URL in the **Reference** field (for example, http://www.OurSite.com).
  - If you are specifying an action to the UI caption of a set of Feature Options, choose the method you want to use to open the URL by selecting from the **Use** dropdown list. (Note that the Feature's Control type must be Selection List to assign an action.)

    To open the same URL regardless of which Option the end user selects, choose **URL** and then specify a location in the **URL** field.

    To allow the end user to open a different URL for each Option, choose **Property Value** and then select a Property from the **Property** dropdown list. (Only Properties common to all Options of the selected Feature appear in the list.) For example, you create a Property called URL and assign it to each Option within a Feature. You then edit the Property for each Option and enter a unique URL as the Property value (for example, http://www.OurSite.com). In the UI module, you assign the Launch URL action to the Feature and select the URL Property. At runtime, a different web page appears depending on which Option label the end user clicks. For information about adding and modifying Properties, see Section 9.14.7, "Properties" on page 9-30.

- **Functional Companion Output**: Select the Component to which the Functional Companion is attached from the **Reference** selection list. Select the Functional Companion from the **Companion** selection list. This type of Functional Companion takes information from Oracle runtime Configurator and provides post processing (for example, custom reporting). See Section 5.14, "Functional Companions" on page 5-67.
- **Functional Companion Auto-Configuration**: Select the Component to which the Functional Companion is attached from the **Reference** selection list. Select the Functional Companion from the **Companion** selection list. This type of Functional Companion queries the state of specific parts of a configuration Model, determines if subsequent parts of the Model should then be changed, and makes the changes to the configuration. (These Functional Companions are sometimes used instead of Logic rules.) See Section 5.14, "Functional Companions" on page 5-67.
- **Update Prices**: Update all dynamic pricing information displayed in the runtime UI (for example, item selling prices or total prices). See Section 9.21.4, "Pricing Display" on page 9-44.

  For information about how a runtime Configurator updates pricing information, see the *Oracle Configurator Implementation Guide*.
- **Go to Home Screen**: Navigate to the first screen in the UI. The first screen depends on the structure of the Model at runtime, which may be affected by effectivity, visibility settings, and so on.
- **Go to Next Screen**: Navigate to the next UI screen according to the structure of the Model at runtime. (The UI and Model structure can change when components become instantiated or if effectivity and visibility settings prevent the display of some screens in the runtime UI.)
- **Go to Previous Screen**: Navigate to the previous UI screen according to the structure of the UI and Model at runtime. (The UI and Model structure can change when components become instantiated or if effectivity and visibility settings prevent the display of some screens in the runtime UI.)
- **Choose Connection**: Display a list of available Models (targets) in the current configuration session to connect the selected component. This is the default action of **Choose Connection** buttons, which Configurator Developer automatically generates for every Connector that exists within the selected Component. See Section 4.3.5, "Creating a Connector" on page 4-6.
- **Activate Instance**: Make all of the UI controls on the selected component's read-only UI screen editable (active). You can assign this action to Button and

Pictures. (This action is relevant only to Models that support updates to installed configurations.)

9.21.6.8 Display

Use this attribute to modify a Value Display object which displays specified data. These are similar to the objects that display the values of Totals and Resources. Instead of referring to a Model node, a **Data Tag** specifies what should be displayed. Select a **Data Tag** from the **Display** dropdown list. The choices are:

- **Total List Price**
- **Total Selling Price**

To specify a different text font for the selected node:

1. Deselect the **Font** box.
2. Click the dialog (**...**) button to open the Font dialog.
3. Choose a font and specify related attributes such as size, style, and so on.

Use the **Borders** dropdown list to specify the type of border to use for the **Value Display**. Choose **Fixed Single** or **None**.

9.21.7 Label

Use this attribute to view or modify default settings for the selected node's **UI caption**. All information in the **Label** attribute is read-only if the selected node is a Reference or it is part of a referenced Model's structure.

The default text of the UI caption depends on how you set the **Generate Captions From** option when creating the UI. See Section 6.2, "Generating a New User Interface" on page 6-16. If the selected node is a BOM Option Class, you can modify the text of the UI caption only if the **Use** setting within the **Option Display** attribute is set to **Label**. See Section 9.21.8, "Option Display" on page 9-50.

Note that if you are using Multiple Language Support (MLS) and you modify the description of a node after generating the UI, Configurator Developer uses the text in the **Text** field at runtime, not the node's description. See Section 1.5, "Multiple Language Support in Oracle Configurator" on page 1-10.

Modify the **Font**, **Background Color**, or **Background Style** if you don't want to use the default settings. The **Background Color** setting applies only if you set **Background Style** to Opaque.

Use the **Align** setting to specify whether text is left-aligned, right-aligned, or centered.

Use the **Action** dropdown list to assign an action to the selected node's UI caption. For example, assign the **Launch URL** action to enable the end user to open a Web page by clicking on the UI caption at runtime. For more information, see Section 6.3.6, "Assigning an Action to Text" on page 6-39.

9.21.8 Option Display

Use this attribute to determine the UI caption (label) for children of the selected Feature or BOM Option Class. Use this attribute to override the **Generate Captions From** setting specified when generating the UI. See Section 6.2, "Generating a New User Interface" on page 6-16.

If the selected node is a BOM Option Class, choose a value from the **Use** dropdown list to specify how to label each Item in the runtime UI. Select **Name**, **Description**, **Name and Description**, or **Label**. If you choose **Label**, you can enter the text that the runtime Configurator uses to label BOM Standard Items at runtime. See Section 9.21.7, "Label" on page 9-49.

If the selected node is a Feature, select the corresponding radio button to **Label each option** or **Display pictures** to show each Option at runtime.

9.21.8.1 Using Labels to Display Options

If you choose to **Label each option**, choose a value from the **Use** list to specify how to label Options in the runtime UI. For example, choose **Name** to display the Option at runtime using the name as it appears in Configurator Developer. Or, choose Name and Description to display the node name and description.

If you choose to label options using both the **Name and Description**, Configurator Developer separates each segment with a comma at runtime. For example:

PT32116, 56K Wireless Modem

The list of display methods depends on the type of UI node selected. For example, if the selected node is:

- A Feature with Options: Choose **Name**, **Description**, **Name and Description**, or **Property**. If you choose **Property**, the Option Display section displays another dropdown list from which you can select a Property (at least one Property must be common to all Options of the selected Feature to label them using this method).

- A BOM Option Class: Choose **Name**, **Description**, **Name and Description**, or **Label**. If you choose **Label**, you can then select each Item's UI node, and then enter a custom UI caption. See Section 9.21.7, "Label" on page 9-49.

If the selected Feature has a Control type of Selection List, you can optionally assign an **Action** to each Option label. This generates a hypertext link for the UI caption and enables the end user to either open a URL in a Web browser or execute a Functional Companion by clicking on it at runtime. For more information about creating hypertext links in the runtime UI, see Section 6.3.6, "Assigning an Action to Text" on page 6-39. For a description of the **Launch URL** and **Functional Companion Output** actions, see Section 9.21.6.7, "Action" on page 9-46.

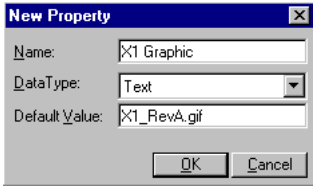
9.21.8.2 Using Pictures to Display Options

Choose **Display pictures** to display Options using a graphic file that you specify. If you choose this setting, Options of the selected Feature must have a Text Property that specifies the name of the image as the Property Value. Otherwise, no values appear in the **Use** dropdown list.

Example: You assign the Property "X1 Graphic" to each Feature Option (in the Model module), then edit this Property for each Option and specify a different image file as the Property value. Then, in the UI module, select the Feature node you want, expand the Option Display attribute, and choose "X1 Graphic" from the **Property** dropdown list. At runtime, Configurator Developer uses the image file you specified to display each Option. Note that to appear in the runtime UI, an image must exist in the OA_MEDIA directory for the Oracle Configurator Servlet. For more information, see the *Oracle Configurator Implementation Guide*.

Figure 9–9 provides an example of how you might define a Text Property to display Feature Options using a picture.

Figure 9–9 New Property Dialog



If the selected Feature has a Control type of Selection List, you can optionally assign an **Action** to each image. This enables the end user to either open a URL in a Web

browser or execute a Functional Companion by clicking on the image at runtime. For a description of the **Launch URL** and **Functional Companion Output** actions, see Section 9.21.6.7, "Action" on page 9-46.

9.21.9 Option Sorting

Use this attribute to specify the order of Feature Options in the runtime UI. You can sort Options alphabetically by label, by their order in the Model structure, or alphanumerically by Property value. You can also choose to display available options first in a list, followed by options that have been excluded from the configuration due to previous end user selections.

See Section 6.3.1.1, "Modifying Default UI Settings" on page 6-22.

9.21.10 Layout

This attribute specifies the position and size of a label or a control field in terms of the number of pixels from the **Left** and **Top** to position the upper left corner of the object, and the **Width** and **Height** in pixels. You can also use the UI Editor to modify the layout of a UI screen by simply dragging and dropping UI controls and labels using the mouse. The values in the Layout attribute are updated automatically when you make changes in the UI Editor. See Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

In some cases, the **Layout** attribute enables you to specify a **Tab Order** for each UI object. This value determines the sequence in which each object in a DHTML UI screen becomes active when an end user presses the **Tab** key at runtime. In most Web browsers, pressing the **Tab** key navigates to each UI control from left to right, and from the top to the bottom of the screen (if you do not specify a **Tab Order**, this is the default behavior).

If you want to customize the navigation sequence, enter a **Tab Order** value for all UI objects in the selected UI screen; otherwise, do not enter a value for any of the UI objects. (A value of 0 (zero) does *not* cause a UI object to be skipped at runtime.)

Note: Refer to Metalink, Oracle's customer support Web site, at `http://metalink.oracle.com` for an Alert about a browser limitation related to the Tab Order functionality.

9.21.11 Visibility

Use this attribute to control whether an option appears in the runtime UI when its logic state changes to Logic False (in other words, excluded from the configuration by the propagation of one or more rules). The values you specify accept or override the **Hide unselectable Controls and Options** setting, which is defined at the UI level. (See Section 6.3.1, "Customizing the User Interface Default Settings" on page 6-21.)

Hide when unselectable determines whether the selected node appears at runtime when its logic state changes to Logic False. You can use this option when the selected UI node is a Feature, BOM Option Class, or a BOM Standard Item.

Hide unselectable Options determines whether to dynamically hide Options of the selected Feature when an Option's logic state changes to Logic False at runtime. For more information about logic states, see Section 5.1.2, "Configuration Rules and Logic State" on page 5-2.

Choose one of the following values:

- **Use Default**: Use the setting defined at the UI level.
- **No**: Display the Option in the runtime UI even when its logic state is Logic False. (By default, Options that are Logic False appear with a red X in the runtime UI.)
- **Yes**: Do not display Options that are Logic False in the runtime UI.

See Section 6.1.5, "Controlling Visibility" on page 6-10.

9.22 The UI Editor

For information about the UI Editor, see Section 6.4, "Previewing Screens with the UI Editor" on page 6-41.

9.22.1 UI Editor Window File Menu

This menu contains only the **Exit**.command, which saves any changes you made and closes the UI Editor.

9.22.2 UI Editor Edit Menu

Table 9–27 shows the commands on the **Edit** menu when you are working in the UI Editor.

Table 9–27 UI Editor Edit Menu Commands

Command	Description
Cut	Cuts the selected node or value. A selected node is displayed in a disabled state until you Paste it.
Copy	Copies the selected node of any type or value.
Paste	Pastes the cut or copied node or value.
Delete	Prompts you to confirm the deletion, then deletes the selected node.
Bring to Front	Sets the z-order of the selected object to the top level.
Send to Back	Sets the z-order of the selected object to the lowest level.
Align	Line up the selected objects along the dimension chosen from the submenu. All selected objects are lined up with the object chosen last. The first three selections, **Lefts**, **Centers**, **Rights**, align the objects by moving them horizontally. The next three selections, **Tops**, **Middles**, **Bottoms**, align the objects by moving them vertically.
Make Same Size	Make the selected objects the same size along the dimension chosen from the submenu, **Width**, **Height**, or **Both**. All selected objects are resized to match the object chosen last.
Horizontal Spacing	Adjust the spacing between selected objects on the horizontal axis. **Make Equal** distributes the objects equally within the width defined by the objects that are farthest away from each other. **Remove** removes all space between the objects, leaving the object closest to the top of the screen unmoved. The order in which objects are selected does not matter.
Vertical Spacing	Adjust the spacing between selected objects on the vertical axis. **Make Equal** distributes the objects equally within the width defined by the objects that are farthest away from each other. **Remove** removes all space between the objects, leaving the object closest to the left of the screen unmoved. The order in which objects are selected does not matter.

9.22.3 Cut, Copy, and Paste

If you delete a UI object, such as a Feature Control, that has an associated Label from the UI Editor, you delete both the object and its Label. The result is the same as if you had deleted it from the UI Tree view. If you delete just the Label, the label text is set to an empty string and the height and width are set to zero. If you cut a label and paste it on another Screen, two things happen. First, the label is deleted from the source screen. Then, a new text object is created on the target screen with the same text, layout, and formatting the label had on the source screen.

9.22.4 Z-Order

The commands **Bring to Front** and **Send to Back** on the **Edit** menu enable you to control the order in which objects that may overlay one another appear on a UI screen. **Bring to Front** causes the selected object to be displayed in front of, or on top of other objects. **Send to Back** causes the selected object to be displayed behind other objects.

Both of these commands operate on multiple selections.

9.23 Test Module

This module launches the test environment specified in the **Test** tab of the Options window (**Tools > Options**). Available test environments include Dynamic HTML in a browser and the Java applet. For more information, see Section 9.9.2.1, "Selecting a Test Environment" on page 9-15.

9.24 The Log Messages Window

You can activate the Log Messages window through the Options selection on the **Tools** menu. See Section 9.9.2, "Options" on page 9-14. For more information on the Log Messages window, see Section 7.2, "Configurator Developer Messages" on page 7-5.

You can also control logging when you run Oracle Configurator Developer using predefined parameters. Starting Configurator Developer using parameters allows you to provide preset values for the mandatory login parameters and bypass the OCD login screen. See the *Oracle Configurator Installation Guide* for details.

There are two menus available in the Log Messages window: **File** and **Settings**.

9.24.1 File Menu

Table 9–28 shows the commands available on the **File** menu in the Log Messages window.

Table 9–28 Log Messages File Menu Commands

Command	Description
Direct To...	Allows the end user to select the disk file to which messages are written. Available only when messages are *not* being written to a file.

Table 9–28 (Cont.) Log Messages File Menu Commands

Command	Description
Close	Stops the recording of messages to a file. Available only when messages are being written to a file.
Keep File Open	By default, the log file is opened and closed with each message to ensure that the message is recorded successfully. When this option is checked, the log file is kept open, which speeds logging slightly, but it raises the risk of losing data if Developer terminates unexpectedly.
Hide	This item is positioned on the menu where 'Close' or 'Exit' is usually found because you cannot actually terminate logging, but you may not want to view the log window. This selection makes the log window invisible. Note that clicking the Close box, or pulling down the control menu and selecting Close also hides the window.

9.24.2 Settings Menu

Table 9–29 shows the commands available on the **Settings** menu in the Log Messages window.

Table 9–29 Log Messages Settings Menu Commands

Command	Description
Report Milliseconds	Ordinarily, log messages are time stamped at one-second intervals; when this option is selected, messages are time stamped to the millisecond.
Report Settings	Sets the minimum severity for reporting messages to the log window and to any open log file. The submenu lists the severity levels in ascending order. DetailTrace messages are the least severe; FATAL are most severe. The default is Notification.
Debug Settings	Sets the minimum severity for debugging messages to the log window and to any open log file. The submenu lists the severity levels in ascending order. DetailTrace messages are the least severe; FATAL are most severe. The default is Notification.
Message Box Text Limit	Allows you to limit the number of characters stored in the message log window. Appending messages to the Log Window may become a performance issue when the window contains about 2500 characters of text. If you select **Unlimited**, note that there is an upper limit of 64Kb. This setting does not cause data to be lost from an open log file. All message written to the window are also written to the file and stored.

A

The Runtime Oracle Configurator

An Oracle Configurator allows you to configure a product by selecting from a list of available options that make up the product. Oracle Configurator ensures that the configured product is valid and orderable by enforcing constraints that govern how all selected items fit together. Oracle Configurator therefore reduces configuration errors, which in turn reduces change order processing and the amount of rework required in manufacturing.

With Oracle Configurator you can:

- Verify product configurations
- Automatically select configuration options

Oracle Configurator supports:

- Assemble-to-Order (ATO) and Pick-to-Order (PTO) BOM Models
- User-defined item attributes
- Flexible configuration constraints
- Integrated configuration validation
- Automatic configuration completion
- Enterprise Resource Planning (ERP) integration
- Customer Relationship Management (CRM) integration

To determine whether a hosting application supports an Oracle Configurator, see the *Oracle Configurator Release Notes*.

A.1 The Oracle Configurator Window

The Oracle Configurator window can be either a Java applet or a DHTML window. The type of Configurator window that appears depends on the hosting application from which it was invoked (usually by clicking a **Configure** button). In Oracle Order Management, Oracle Bills of Material, Order Capture, and Oracle TeleSales you can configure an item in either a DHTML window or a Java applet. In *i*Store, Sales for Comms, Quoting, and Oracle Sales Online you can configure an item only in a DHTML window.

A Configurator Developer user chooses a UI style when generating a User Interface. (See Section 6.2, "Generating a New User Interface" on page 6-16.) When the Model and UI are ready for testing, a Configurator Developer makes them available to specific hosting applications by publishing them. (See Section 2.2, "Publishing" on page 2-11.)

The layout and method of configuring a product vary depending on which type of user interface you see.

In the Java applet:

- Navigate from one component to another using the navigation tree that appears in the left-hand region in the model structure pane. This tree displays all configurable components for the selected product. Components that contain required selections are marked by a red asterisk in the navigation tree.
- Select a component in the navigation tree to view the available features and options of that component in the selection region (upper-right).
- Configure each component by selecting options in the selection region.
- View information about all items added to the configuration in the summary region (bottom-right).
- Click **Done** to save the configuration and return to the hosting application.

In the DHTML window:

- Some needs-assessment questions may appear to gather information about how you will use the product. Your answers typically default some options and exclude others from the configuration.
- Navigate to the next component to be configured using either the navigation tree or navigation buttons provided in each screen. Components that contain required selections are marked by a red asterisk in the navigation tree. (Note that the navigation tree is optional in the DHTML window and might not appear.)

- Configure each component on a separate screen.
- Click **Summary** to view information about all items added to the configuration. Click the Configuration button to continue.
- Click **Done** to save the configuration and return to the hosting application.

Implementors set values for specific profile options to determine how Oracle Applications access the Oracle Configurator window. For information about the profile options that affect Oracle Configurator, see the *Oracle Configurator Installation Guide*.

A.1.1 Configuring an Item

Oracle Configurator validates each selection against the rules defined for the item that you are configuring. Your selections can trigger configuration rules that automatically exclude or make available other options in the product. You may see this occur on the current screen when you select an option.

If you make a selection that violates a configuration rule, a message describes the violation and provides suggestions on how to proceed.

To configure an item:

1. Begin selecting from the list of available options, or answer any needs-assessment questions that appear.

   For more information about selecting options in a Configurator window, see Table A–1 on page A-4.

2. When you finish making selections for one component, navigate to the other components until you configure the entire product.
3. Review a summary of the configuration in either the Summary screen (DHTML) or the lower region of the window (Java applet). This section displays information about all options added to the configuration, such as quantity, price, and the total price for the configured product.
4. If an **Availability** button is displayed, click it if you want to see the Available To Promise (ATP) dates for the relevant items.
5. When you are finished, click **Done** to save the configuration and continue processing your order.

   If the configuration is satisfied, you have made all required and valid selections.

A message appears if the configuration is unsatisfied. In this case you can either ignore the message and continue, or complete the configuration by selecting additional options.

Warning: Ending a configuration session using a method other than clicking Done causes all selections to be lost (for example, clicking Cancel or choosing File > Close from the browser window).

6. The host application closes the Configurator window and continues processing your order.

Table A–1 Selecting Options in an Oracle Configurator Window

To select this . . .	Do this . . .
One option	Select the check box for each desired option.
One option in a group of mutually exclusive options	Select the check box for the desired option. (This list might be presented as a set of radio buttons.)
One or more options in a group that allows multiple selections	Select the check boxes for all desired options.
A quantity for a numeric option	Enter the desired quantity. Entering a quantity greater than zero automatically selects the option.

A.1.2 Configuring an Order from a Bill of Materials

In Oracle Order Management, you can configure products created from a Bill of Materials in Oracle Bills of Material (BOM) when:

To learn how to configure an order, see Section A.1.1 on page A-3.

- You have not installed Oracle Configurator. In this case, you select options from an ATO or PTO BOM Model using the Order Management Options window.
- Oracle Configurator is installed, but no configuration model exists for the selected ATO or PTO BOM Model. In this case, you simply make selections in the Java applet from the BOM as it was defined in Oracle Bills of Material.
- There is no Oracle Configurator servlet running. In this case, you configure the item by selecting options from the Order Management Options window.

- The relevant Oracle Applications profile options are set appropriately. For more information, see the *Oracle Configurator Installation Guide*.

A.1.3 Preconfiguring an Item

You can create a configured bill of material for a pre-defined ATO item by invoking Oracle Configurator from Oracle Bills of Material. This is useful when the same configuration of an item is ordered frequently.

For more information about preconfiguring items, see the *Oracle Bills of Material User's Guide*.

A.1.4 Keyboard Access in the Oracle Configurator Window

Oracle Configurator enables end users with disabilities to navigate the Configurator window using only the keyboard. For example, end users can navigate to each option in the Configurator window by pressing the **Tab** key. Typically, keyboard UI navigation travels from left to right and from the top to the bottom of the page, and each option is highlighted when it can be selected.

Table A–2 on page A-5 lists the available keystrokes and the corresponding actions in the Configurator window. These commands perform the same action in both the Configurator Java applet and DHTML User Interfaces.

Table A–2 Keyboard Access in the Configurator Window

Press this. . .	If you want to . . .
Tab	Move the focus forward to each option in the UI (from left to right, top to bottom of the page).
	Navigate to the next frame. For example, from the navigation tree to the selection window.
Shift+Tab	Move the focus to each option in the reverse order through the UI (right to left, bottom to top).
	Navigate to the previous frame.
Enter	Select an option (that is, add it to the configuration) or execute an action (for example, activate the **Done** button to commit the configuration).
	Expand or collapse the selected branch in the navigation tree.
Space Bar	Toggle the state of a check box. For example, change the status from selected (true) to deselected (false).

Table A–2 Keyboard Access in the Configurator Window

Press this. . .	If you want to . . .
Up and Down Arrow Keys	Move the focus through each option in a list.
	Move the focus up and down through the navigation tree.
Left and Right Arrow Keys	Scroll to the left or right.
Delete	Deselect a selected option.

A.2 Creating Instances at Runtime

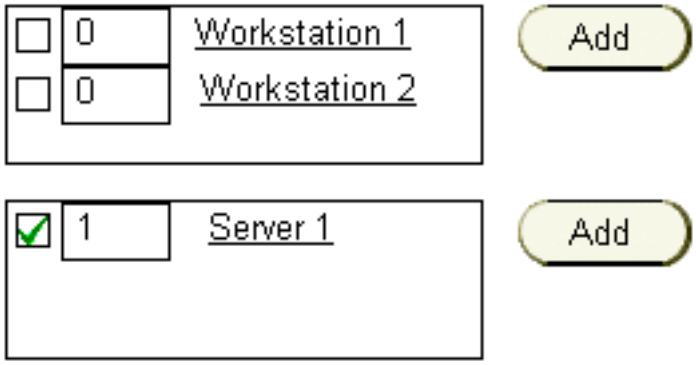
At runtime, an **Add** button appears in a component's parent UI screen if the component's **Instances Minimum** and **Maximum** values are not equal in Configurator Developer. You click this button to add instances of the component to the configuration. Each component name appears as a link that you can click to navigate to the component's UI screen.

Note: The ability to instantiate components multiple times in a runtime Oracle Configurator is available only in a DHTML User Interface. You cannot instantiate components multiple times in a Java applet UI.

Figure A–1 shows components that can be instantiated multiple times and the buttons that enable you to add new instances of each component. The Server component is selected by default, which indicates that it is required in the configuration (that is, its **Instances Minimum** value is set to 1 in Configurator Developer). The number in the input field corresponds to the Default Quantity value defined in Oracle Bills of Material.

Note: If multiple instances of a component exist in the configuration, the **Description** column in the Summary screen displays the instance name instead of the BOM item's description. However, it is possible to display instance names in a separate column. For more information, see the section on the CZ_DB_SETTINGS table in the *Oracle Configurator Implementation Guide*.

Figure A–1 Instantiating a Component at Runtime



Each time you add an instance, the runtime Oracle Configurator appends a number to the newly instantiated component's name. Note that if you delete a component by clicking the **Delete** button, the instance numbers may no longer appear sequentially in the runtime UI.

If the instance being configured is not complete and you click the **Add** button, then Oracle Configurator creates the new instance (in other words, you do not have to complete the selected instance before creating another). Note that you can create new instances only if the maximum number of instances allowed in the configuration has not yet been reached (this value is defined in Configurator Developer).

When you save a configuration that is valid but has incomplete components, Oracle Configurator displays a message listing the incomplete items. However, you may choose to continue and pass the saved configuration back to Oracle Order Management. When you restore a saved configuration in which some instances were left in an incomplete state, those instances are preserved in the restored configuration.

Because configured items with incomplete components cannot be manufactured, the batch validation process displays errors when an Oracle Order Management user tries to book the order.

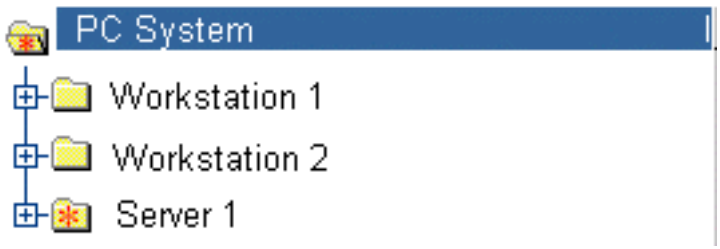
For more information about **Add** and **Delete** buttons, see Section 6.1.3.3 on page 6-8.

A.2.1 The Navigation Tree View

Figure A–2 shows the runtime Navigation Tree for a Model containing instantiable components when a configuration session begins. In Oracle Configurator Developer, the **Instances Minimum** for the Workstation PTO Model is set to 2, so two instances of Workstation exist when the configuration session begins. As in the runtime UI, each component in the Navigation Tree has an instance number appended to the name, and a red asterisk appears next to an instance if it contains required selections.

Oracle Configurator updates the Navigation Tree whenever you add a new instance to the configuration. A Configurator Developer user can customize the naming convention used in the Navigation Tree by implementing a Functional Companion. For more information see the *Oracle Configurator Implementation Guide*.

Figure A–2 Runtime Navigation Tree with Instantiable Components



The Navigation Tree view is updated but not reordered when new instances are added to the configuration.

A.2.2 Behavior of Instances

Effective dates and Usages are defined on and affect whole components, not separate or specific instances of the elements. If the instantiable component is ineffective, then all of the components with it are ineffective. See Section 3.3, "Effectivity" on page 3-4 for additional information.

Quantity Cascade of BOM items is preserved within any configuration of an instance. For more information about Quantity Cascade, see Section 3.4.2, "Imported BOM Rules" on page 3-9.

You can change the quantity of a specific component in the Oracle Configurator window and on the line item for that component in Oracle Order Management. The

number of component instances that exist in a configuration can only be modified in the Oracle Configurator window.

Pricing and Available To Promise (ATP) information are done per instance. For more information about pricing and ATP, see the *Oracle Configurator Implementation Guide*.

Glossary of Terms and Acronyms

This glossary contains definitions that you may need while working with Oracle Configurator.

Active Model

The compiled structure and rules of a **configuration model** that is loaded into memory on the Web server at **configuration session** initialization and used by the **Oracle Configurator engine** to validate runtime selections. The Active Model must be generated either in **Oracle Configurator Developer** or programmatically in order to access the configuration model at **runtime**.

API

Application Programming Interface

applet

A Java application running inside a Web browser. *See also* **Java** and **servlet**.

application architecture

The software structure of an application at **runtime**. Architecture affects how an application is used, maintained, extended, and changed.

architecture

See **application architecture**.

ATO

Assemble to Order

ATP

Available to Promise

attribute

The defining characteristic of something. Models have attributes such as Effectivity Sets and Usage. Components, Features, and Options have attributes such as Name, Description, and Effectivity.

benchmark

Represents performance data collected during **runtime** tests under various conditions that emulate expected and extreme use of the product.

beta

An external release, delivered as an installable application, and subject to acceptance, **validation**, and **integration testing**. Specially selected and prepared **end users** may participate in beta testing.

bill of material

A list of Items associated with a parent Item, such as an assembly, and information about how each Item relates to that parent Item.

Bills of Material

The application in Oracle Applications in which you define a **bill of material**.

BOM

See **bill of material**.

BOM item

The **node** imported into **Oracle Configurator Developer** that corresponds to an Oracle **Bills of Material** item. Can be a **BOM Model**, **BOM Option Class node**, or **BOM Standard Item node**.

BOM Model

A model that you import from Oracle **Bills of Material** into **Oracle Configurator Developer**. When you import a BOM Model, effective dates, **ATO** rules, and other data are also imported into Configurator Developer. In Configurator Developer, you can extend the structure of the BOM Model, but you cannot modify the BOM Model itself or any of its **attributes**.

BOM Model node

The imported **node** in **Oracle Configurator Developer** that corresponds to a **BOM Model** created in Oracle **Bills of Material**.

BOM Option Class node

The imported **node** in **Oracle Configurator Developer** that corresponds to a BOM Option Class created in Oracle **Bills of Material**.

BOM Standard Item node

The imported **node** in **Oracle Configurator Developer** that corresponds to a BOM Standard Item created in Oracle **Bills of Material**.

Boolean Feature

An **element** of a **component** in the **Model** that has two **option**s: true or false.

bug

See **defect**.

build

A specific **instance** of an application during its construction. A build must have an install program early in the project so that application **implementer**s can **unit test** their latest work in the context of the entire available application.

CIO

See **Oracle Configuration Interface Object (CIO)**.

client

A **runtime** program using a **server** to access functionality shared with other clients.

Comparison rule

An **Oracle Configurator Developer** rule type that establishes a relationship to determine the selection state of a logical **Item** (Option, Boolean Feature, or List-of-Options Feature) based on a comparison of two numeric values (numeric **Feature**s, **Total**s, **Resource**s, **Option** counts, or numeric constants). The numeric values being compared can be computed or they can be discrete intervals in a continuous numeric input.

Compatibility rule

An **Oracle Configurator Developer** rule type that establishes a relationship among **Feature**s in the Model to control the allowable combinations of **Option**s. *See also*, **Property-based Compatibility rule**.

Compatibility Table

A kind of Explicit Compatibility rule. For example, a type of compatibility relationship where the allowable combination of **Option**s are explicitly enumerated.

component

A piece of something or a configurable element in a **model** such as a **BOM Model**, **Model**, or **Component**.

Component

An element of the **model structure**, typically containing **Feature**s, that is configurable and instantiable. An **Oracle Configurator Developer** node type that represents a configurable element of a **Model**. Corresponds to one UI screen of selections in a runtime **Oracle Configurator**.

Component Set

An element of the **Model** that contains a number of instantiated **Component**s of the same type, where each Component of the set is independently configured.

concurrent manager

A process manager that coordinates the **concurrent process**es generated by **user**s' **concurrent request**s. An Oracle Applications product group can have several concurrent managers.

concurrent process

A task that can be scheduled and is run by a **concurrent manager**. A concurrent process runs simultaneously with interactive functions and other concurrent processes.

concurrent processing facility

An Oracle Applications facility that runs time-consuming, non-interactive tasks in the background.

concurrent program

Executable code (usually written in SQL*Plus or Pro*C) that performs the function(s) of a requested task. Concurrent programs are stored procedures that

perform actions such as generating reports and copying data to and from a database.

concurrent request

A user-initiated request issued to the concurrent processing facility to submit a non-interactive task, such as running a report.

configuration

A specific set of specifications for a product, resulting from selections made in a runtime **configurator**.

configuration attribute

A characteristic of an **item** that is defined in the **host application** (outside of its inventory of items), in the **Model**, or captured during a **configuration session**. Configuration attributes are inputs from or outputs to the host application at initialization and termination of the configuration session, respectively.

Configuration Interface Object

See **Oracle Configuration Interface Object (CIO)**.

configuration model

Represents all possible configurations of the available **option**s, and consists of **model structure** and **rules**. It also commonly includes **User Interface** definitions and **Functional Companion**s. A configuration model is usually accessed in a **runtime Oracle Configurator window**. *See also* **model**.

configuration rules

The **Oracle Configurator Developer Logic rules**, **Compatibility rule**s, **Comparison rule**s, **Numeric rule**s, and **Design Chart**s available for defining **configuration**s. *See also* **rules**.

configuration session

The time from launching or invoking to exiting **Oracle Configurator**, during which **end user**s make selections to configure an orderable product. A configuration session is limited to one **configuration model** that is loaded when the session is initialized.

configurator

The part of an application that provides custom configuration capabilities. Commonly, a window that can be launched from a hosting application so **end user**s

can make selections resulting in valid **configuration**s. *Compare* **Oracle Configurator**.

connectivity

The connection between **client** and database **server** that allows data communication.

The connection across components of a model that allows modeling such products as networks and material processing systems.

Connector

The **node** in the **model structure** that enables an **end user** at **runtime** to connect the Connector node's parent to a referenced **Model**.

Container Model

A type of **BOM Model** that you import from Oracle **Bills of Material** into **Oracle Configurator Developer** to create configuration models containing **connectivity** and trackable components. Configurations created from Container Models can be tracked and updated in Oracle Install Base

context

The surrounding text or conditions of something.

Determines which context-sensitive segments of a flexfield in the Oracle Applications database are available to an application or **user**. Used in defining **configuration attribute**s.

Contributes to

A relation used to create a specific type of **Numeric rule** that accumulates a total value. *See also* **Total**.

Consumes from

A relation used to create a specific type of **Numeric rule** that decrementing a total value, such as specifying the quantity of a **Resource** used.

count

The number or quantity of something, such as selected **option**s. *Compare* **instance**.

CRM

See **Customer Relationship Management**

CTO

Configure to Order

customer

The person for whom products are configured by **end users** of the **Oracle Configurator** or other **ERP** and **CRM** applications. Also the end users themselves directly accessing **Oracle Configurator** in a Web store or kiosk.

customer-centric extensions

See **customer-centric views**.

customer-centric views

Optional extensions to core functionality that supplement configuration activities with **rules** for **preselection**, **validation**, and **intelligent views**. View capabilities include generative geometry, drawings, sketches and schematics, charts, performance analyses, and **ROI** calculations.

Customer Relationship Management

The aspect of the enterprise that involves contact with customers, from lead generation to support services.

customer requirements

The needs of the customer that serve as the basis for determining the configuration of products, **systems**, and services. Also called needs assessment. *See* **guided buying or selling**.

CZ

The product shortname for **Oracle Configurator** in Oracle Applications.

data import

Populating the **Oracle Configurator schema** with enterprise data from **ERP** or legacy systems via **import tables**.

Data Integration Object

Also known as the DIO, the Data Integration Object is a **server** in the **runtime** application that creates and manages the interface between the **client** (usually a **user interface**) and the **Oracle Configurator schema**.

data maintenance environment

The environment in which the runtime **Oracle Configurator** data is maintained.

data source

A programmatic reference to a database. Referred to by a data source name (DSN).

DBMS

Database Management System

default

A predefined value. In a **configuration**, the automatic selection of an **option** based on the **preselection** rules or the selection of another option.

Defaults rule

An **Oracle Configurator Developer** Logic rule that determines the logic state of **Feature**s or **Option**s in a default relation to other Features and Options. For example, if A Defaults B, and you select A, B becomes Logic True (selected) if it is available (not Logic False).

defect

A failure in a product to satisfy the **user**s' requirements. Defects are prioritized as critical, major, or minor, and fixes range from corrections or workarounds to enhancements. Also known as a bug.

defect tracking

A system of identifying defects for managing additional tests, testing, and approval for release to **user**s.

deliverable

A work product that is specified for review and delivery.

demonstration

A presentation of the tested application, showing a particular usage scenario.

Design Chart

An **Oracle Configurator Developer** rule type for defining advanced Explicit Compatibilities interactively in a table view.

design review

A technical review that focuses on application or **system** design.

developer

The person who uses **Oracle Configurator Developer** to create a **configurator**. *See also* **implementer** and **user**.

Developer

The tool (**Oracle Configurator Developer**) used to create **configuration models**.

DHTML

Dynamic Hypertext Markup Language

DIO

See **Data Integration Object**.

distributed computing

Running various **components** of a **system** on separate machines in one network, such as the database on a database **server** machine and the application software on a Web server machine.

DLL

Dynamically Linked Library

DSN

See **data source**.

element

Any entity within a **model**, such as **Options**, **Totals**, **Resources**, UI controls, and **components**.

end user

The ultimate user of the runtime **Oracle Configurator**. The types of end users vary by project but may include salespeople or distributors, administrative office staff, marketing personnel, order entry personnel, product engineers, or customers directly accessing the application via a Web browser or kiosk. *Compare* **user**.

enterprise

The **systems** and **resources** of a business.

environment

The arena in which software tools are used, such as operating system, applications, and **server** processes.

ERP

Enterprise Resource Planning. A software system and process that provides automation for the customer's back-room operations, including order processing.

Excludes rule

An **Oracle Configurator Developer** Logic rule determines the logic state of **Feature**s or **Option**s in an excluding relation to other Features and Options. For example, if A Excludes B, and if you select A, B becomes Logic False, since it is not allowed when A is true (either User or Logic True). If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or **Unknown**. *See* **Negates rule**.

extended functionality

A release after delivery of core functionality that extends that core functionality with **customer-centric views**, more complex proposal generation, discounting, quoting, and expanded integration with **ERP**, **CRM**, and third-party software.

feature

A characteristic of something, or a configurable element of a **component** at **runtime**.

Feature

An element of the **model structure**. Features can either have a value (numeric or Boolean) or enumerated **Option**s.

Functional Companion

An extension to the **configuration model** beyond what can be implemented in Configurator Developer.

An object associated with a **Component** that supplies methods that can be used to initialize, validate, and generate **customer-centric views** and outputs for the **configuration**.

functional specification

Document describing the functionality of the application based on **user** requirements.

guided buying or selling

Needs assessment questions in the **runtime** UI to guide and facilitate the configuration process. Also, the **model structure** that defines these questions. Typically, guided selling questions trigger **configuration rules** that automatically select some product **option**s and exclude others based on the **end user's** responses.

host application

An application within which **Oracle Configurator** is embedded as integrated functionality, such as Order Management or *i*Store.

HTML

Hypertext Markup Language

ICX

Inter-Cartridge Exchange

implementation

The stage in a project between defining the problem by selecting a configuration technology vendor, such as Oracle, and deploying the completed configuration application. The implementation stage includes gathering requirements, defining test cases, designing the application, constructing and testing the application, and delivering it to **end user**s. *See also* **developer** and **user**.

implementer

The person who uses **Oracle Configurator Developer** to build the **model structure**, rules, and UI customizations that make up a **runtime** Oracle Configurator. Commonly also responsible for enabling the integration of **Oracle Configurator** in a **host application**.

Implies rule

An **Oracle Configurator Developer** Logic rule that determines the logic state of **Feature**s or **Option**s in an implied relation to other Features and Options. For example, if A Implies B, and you select A, B becomes Logic True. If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or **Unknown**. *See* **Requires rule**.

import server

A database **instance** that serves as a source of data for **Oracle Configurator**'s Populate, Refresh, and Synchronization concurrent processes. The import server is sometimes referred to as the remote server.

import tables

Tables mirroring the Oracle Configurator schema Item Master structure, but without integrity constraints. Import tables allow batch population of the Oracle Configurator schema's Item Master. Import tables also store extractions from Oracle Applications or **legacy data** that create, update, or delete records in the Oracle Configurator schema **Item Master**.

incremental construction

The process of organizing the construction of the application into **build**s, where each build is designed to meet a specified portion of the overall requirements and is **unit test**ed.

initialization message

The **XML** message sent from a **host application** to the **Oracle Configurator Servlet**, containing data needed to initialize the runtime Oracle Configurator. *See also* **termination message**.

install program

Software that sets up the local machine and installs the application for testing and use.

Instance

An **Oracle Configurator Developer** attribute of a **component's node** that specifies a minimum and maximum value. *See also* **instance**.

instance

A **runtime** occurrence of a **component** in a configuration. *See also* **instantiate**. *Compare* **count**.

Also, the memory and processes of a database.

instantiate

To create an instance of something. Commonly, to create an **instance** of a **component** in the runtime **user interface** of a **configuration model**.

integration

The process of combining multiple software **component**s and making them work together.

integration testing

Testing the interaction among software programs that have been integrated into an application or **system**. *Compare* **unit test**.

intelligent views

Configuration output, such as reports, graphs, schematics, and diagrams, that help to illustrate the value proposition of what is being sold.

IS

Information Services

item

A product or part of a product that is in inventory and can be delivered to customers.

Item

A Model or part of a Model that is defined in the **Item Master**. Also data defined in Oracle Inventory.

Item Master

Data stored to structure the Model. Data in the Item Master is either entered manually in **Oracle Configurator Developer** or imported from Oracle Applications or a legacy system.

Item Type

Data used to classify the Items in the Item Master. Item Catalogs imported from Oracle Inventory are Item Types in **Oracle Configurator Developer**.

Java

An object-oriented programming language commonly used in internet applications, where Java applications run inside Web browsers and **server**s. *See also* **applet** and **servlet**.

LAN

Local Area Network

LCE

Logical Configuration Engine. *Compare* **Active Model**.

legacy data

Data that cannot be imported without creating custom extraction programs.

load

Storing the **configuration model** data in the **Oracle Configurator Servlet** on the Web server. Also, the time it takes to initialize and display a configuration model if it is not preloaded.

The burden of transactions on a **system**, commonly caused by the ratio of **user** connections to CPUs or available memory.

log file

A file containing errors, warnings, and other information that is output by the running application.

Logic rules

Logic rules directly or indirectly set the logical state (User or Logic True, User or Logic False, or **Unknown**) of **Feature**s and **Option**s in the Model.

There are four primary Logic rule relations: Implies, Requires, Excludes, and Negates. Each of these rules takes a list of Features or Options as operands. *See also* **Implies rule**, **Requires rule**, **Excludes rule**, and **Negates rule**.

maintainability

The characteristic of a product or process to allow straightforward **maintenance**, alteration, and extension. Maintainability must be built into the product or process from inception.

maintenance

The effort of keeping a **system** running once it has been deployed, through **defect** fixes, procedure changes, infrastructure adjustments, data replication schedules, and so on.

Metalink

Oracle's technical support Web site at:

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

Model

The entire hierarchical "tree" view of all the data required for **configuration**s, including **model structure**, variables such as **Resource**s and **Total**s, and elements in

support of intermediary rules. Includes both imported **BOM Models** and Models created in Configurator Developer. May consist of BOM Option Classes and BOM Standard Items.

model

A generic term for data representing products. A model contains **elements** that correspond to **items**. Elements may be **components** of other objects used to define products. A **configuration model** is a specific kind of model whose elements can be configured by accessing an **Oracle Configurator window**.

model-driven UI

The graphical views of the **model structure** and rules generated by **Oracle Configurator Developer** to present **end users** with interactive product selection based on **configuration models**.

model structure

Hierarchical "tree" view of data composed of **elements** (**Models**, **Components**, **Features**, **Options**, **BOM Models**, **BOM Option Class nodees**, **BOM Standard Item nodes**, **Resources**, and **Totals**). May include reusable **components** (**References**).

MS

Microsoft Corporation

Negates rule

A type of **Oracle Configurator Developer** Logic rule that determines the logic state of **Features** or **Options** in a negating relation to other Features and Options. For example, if one **option** in the relationship is selected, the other option must be Logic False (not selected). Similarly, if you deselect one option in the relationship, the other option must be Logic True (selected). *See* **Excludes rule**.

node

The icon or location in a **Model** tree in **Oracle Configurator Developer** that represents a **Component**, **Feature**, **Option** or variable (**Total** or **Resource**), **Connector**, **Reference**, **BOM Model**, **BOM Option Class node**, or **BOM Standard Item node**.

Numeric rule

An **Oracle Configurator Developer** rule type that express constraint among model elements in terms of numeric relationships. *See also*, **Contributes to** and **Consumes from**.

OC

See **Oracle Configurator**.

ODBC

Open Database Connectivity. A database access method that uses drivers to translate an application's data queries into **DBMS** commands.

OCD

See **Oracle Configurator Developer**.

opportunity

The workspace in Oracle Sales Online in which products, **systems**, and services are configured, quotes and proposals are generated, and orders are submitted.

option

A logical selection made by the **end user** when configuring a **component**.

Option

An element of the **Model**. A choice for the value of an enumerated **Feature**.

Oracle Configuration Interface Object (CIO)

A **server** in the **runtime** application that creates and manages the interface between the **client** (usually a **user interface**) and the underlying representation of **model structure** and rules in the **Active Model**.

The CIO is the **API** that supports creating and navigating the Model, querying and modifying selection states, and saving and restoring **configurations**.

Oracle Configurator

The product consisting of development tools and **runtime** applications such as the **Oracle Configurator schema**, **Oracle Configurator Developer**, and runtime Oracle Configurator. Also the runtime Oracle Configurator variously packaged for use in networked or Web deployments.

Oracle Configurator architecture

The three-tier **runtime** architecture consists of the **User Interface**, the **Active Model**, and the **Oracle Configurator schema**. The application development architecture consists of **Oracle Configurator Developer** and the Oracle Configurator schema, with test instances of a runtime **Oracle Configurator**.

Oracle Configurator Developer

The suite of tools in the **Oracle Configurator** product for constructing and maintaining **configurator**s.

Oracle Configurator engine

Also **LCE**. *Compare* **Active Model**.

Oracle Configurator schema

The implementation version of the standard runtime **Oracle Configurator** data-warehousing schema that manages data for the **configuration model**. The implementation schema includes all the data required for the **runtime** system, as well as specific tables used during the construction of the **configurator**.

Oracle Configurator Servlet

Vehicle for **Oracle Configurator** containing the UI Server.

Oracle Configurator window

The **user interface** that is launched by accessing a **configuration model** and used by **end user**s to make the selections of a **configuration**.

Oracle SellingPoint Application

No longer available or supported.

output

The output generated by a **configurator**, such as quotes, proposals, and **customer-centric views**.

performance

The operation of a product, measured in throughput and other data.

Populator

An entity in **Oracle Configurator Developer** that creates **Component**, **Feature**, and **Option node**s from information in the **Item Master**.

preselection

The default state in a **configurator** that defines an initial selection of **Component**s, **Feature**s, and **Option**s for configuration.

A process that is implemented to select the initial element(s) of the **configuration**.

product

Whatever is ordered and delivered to customers, such as the output of having configured something based on a model. Products include intangible entities such as services or contracts.

project manager

A member of the project team who is responsible for directing the project during implementation.

project plan

A document that outlines the logistics of successfully implementing the project, including the schedule.

Property

A named value associated with a **node** in the **Model** or the **Item Master**. A set of Properties may be associated with an Item Type. After importing a BOM Model, Oracle Inventory Catalog Descriptive Elements are Properties in **Oracle Configurator Developer**.

Property-based Compatibility rule

A kind of compatibility relationship where the allowable combinations of **Option**s are specified implicitly by relationships among Property values of the Options.

prototype

A construction technique in which a preliminary version of the application, or part of the application, is built to facilitate **user** feedback, prove feasibility, or examine other implementation issues.

PTO

Pick to Order

publication

A unique deployment of a **configuration model** (and optionally a **user interface**) that enables a developer to control its availability from hosting applications such as Oracle Order Management or *i*Store. Multiple publications can exist for the same configuration model, but each publication corresponds to only one **Model** and **User Interface**.

publishing

The process of creating a **publication** record in **Oracle Configurator Developer**, which includes specifying applicability parameters to control **runtime** availability and running an Oracle Applications concurrent process to copy data to a specific database.

QA

Quality Assurance

RAD

Rapid Application Development

RDBMS

Relational Database Management System

reference

The ability to reuse an existing **Model** or **Component** within the structure of another Model (for example, as a subassembly).

Reference

An **Oracle Configurator Developer** node type that denotes a **reference** to another **Model**.

regression test

An automated test that ensures the newest **build** still meets previously tested requirements and functionality. *See also* **incremental construction**.

Requires rule

An **Oracle Configurator Developer** Logic rule that determines the logic state of **Feature**s or **Option**s in a requirement relation to other Features and Options. For example, if A Requires B, and if you select A, B is set to Logic True (selected). Similarly, if you deselect A, B is set to Logic False (deselected). See **Implies rule**.

resource

Staff or equipment available or needed within an enterprise.

Resource

A variable in the **Model** used to keep track of a quantity or supply, such as the amount of memory in a computer. The value of a Resource can be positive or zero,

and can have an Initial Value setting. An error message appears at **runtime** when the value of a Resource becomes negative, which indicates it has been over-consumed. Use **Numeric rules** to contribute to and consume from a Resource.

Also a specific node type in **Oracle Configurator Developer**. *See also* **node**.

reusable component

See **reference** and **model structure**.

reusability

The extent to and ease with which parts of a **system** can be put to use in other systems.

RFQ

Request for Quote

ROI

Return on Investment

rules

Also called business rules or **configuration rules**. Constraints applied among elements of the product to ensure that defined relationships are preserved during configuration. Elements of the product are **Components**, **Features**, and **Options**. Rules express logic, numeric parameters, implicit compatibility, or explicit compatibility. Rules provide **preselection** and **validation** capability in **Oracle Configurator**.

See also **Comparison rule**, **Compatibility rule**, **Design Chart**, **Logic rules** and **Numeric rule**.

runtime

The environment and context in which applications are run, tested, or used, rather than developed.

The environment in which an **implementer** (tester), **end user**, or **customer** configures a product whose model was developed in **Oracle Configurator Developer**. *See also* **configuration session**.

sales configuration

A part of the sales process to which configuration technology has been applied in order to increase sales effectiveness and decrease order errors. Commonly identifies **customer requirements** and product configuration.

schema

The tables and objects of a data model that serve a particular product or business process. *See* **Oracle Configurator schema**.

SCM

Supply Chain Management

server

Centrally located software processes or hardware, shared by **client**s.

servlet

A Java application running inside a Web server. *See also* **Java**, **applet**, and **Oracle Configurator Servlet**.

SFA

Sales Force Automation

solution

The deployed **system** as a response to a problem or problems.

SQA

Software Quality Assurance

SQL

Structured Query Language

system

The hardware and software **component**s and infrastructure integrated to satisfy functional and **performance** requirements.

termination message

The **XML** message sent from the **Oracle Configurator Servlet** to a **host application** after a **configuration session**, containing configuration outputs. *See also* **initialization message**.

test case

A description of inputs, execution instructions, and expected results that are created to determine whether a specific software feature works correctly or a specific requirement has been met.

Total

A variable in the **Model** used to accumulate a numeric total, such as total price or total weight.

Also a specific node type in **Oracle Configurator Developer**. *See also* **node**.

UI

See **User Interface**.

Unknown

The logic state that is neither true nor false, but unknown at the time a **configuration session** begins or when a Logic rule is executed. This logic state is also referred to as Available, especially when considered from the point of view of the **runtime Oracle Configurator end user**.

unit test

Execution of individual routines and modules by the application **implementer** or by an independent test consultant to find and resolve **defect**s in the application. *Compare* **integration testing**.

update

Moving to a new version of something, independent of software release. For instance, moving a production **configurator** to a new version of a **configuration model**, or changing a **configuration** independent of a model **update**.

upgrade

Moving to a new release of **Oracle Configurator** or **Oracle Configurator Developer**.

user

The person using a product or system. Used to describe the person using **Oracle Configurator Developer** tools and methods to build a **runtime Oracle Configurator**. *Compare* **end user**.

User Interface

The part of **Oracle Configurator architecture runtime** architecture that is generated from the **model structure** and provides the graphical views necessary to create **configuration**s interactively. Interacts with the **Active Model** and data to give **end user**s access to customer requirements gathering, product selection, and **customer-centric views**.

user interface

The visible part of the application, including menus, dialog boxes, and other on-screen elements. The part of a **system** where the **user** interacts with the software. Not necessarily generated in **Oracle Configurator Developer**.

user requirements

A description of what the **configurator** is expected to do from the **end user's** perspective.

user's guide

Documentation on using the application or **configurator** to solve the intended problem.

validation

Tests that ensure that configured **component**s will meet specific criteria set by an enterprise, such as that the components can be ordered or manufactured.

Validation

A type of **Functional Companion** that is implemented to ensure that the configured **component**s will meet specific criteria.

VAR

Value-Added Reseller

variable

Parts of the **Model** that are represented by **Total**s, **Resource**s, or numeric **Feature**s.

VB

Microsoft Visual Basic. Programming language in which portions of **Oracle Configurator Developer** are written.

verification

Tests that check whether the result agrees with the specification.

WAN

Wide Area Network

Web

The portion of the Internet that is the World Wide Web.

WIP

Work In Progress

XML

Extensible Markup Language, a highly flexible markup language for transferring data between **Web** applications. Used for the **initialization message** and **termination message** of the **Oracle Configurator Servlet**.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z