Siebel Product Administration Guide

Siebel 2018

April 2018

ORACLE

Contents

Chapter 8: Smart Part Numbers for Products with Attributes

Chapter 9: Designing Products with Components

Chapter 10: Managing Products with Components

Chapter 11: Creating Custom Siebel Configurator User Interfaces

Chapter 12: Siebel Configurator UI Properties

Chapter 13: Siebel Configurator Web Templates

Chapter 14: Configuration Constraints

Chapter 15: Configuration Links

Chapter 16: Configuration Resources

Chapter 17: Configuration Constraint Template Reference

Chapter 18: Siebel Configurator Rule Assembly Language

Chapter 19: Siebel Configurator Scripts

Chapter 22: Product and Promotion Eligibility and Compatibility

Chapter 23: Creating Validation Rules for Customizable Products

Chapter 24: Siebel Configurator Technical Reference

Chapter 25: Configurator Workflow and Method Reference

Chapter 26: Siebel Configurator API Reference

Chapter 27: Siebel Configurator Version 6.x, 7.0 and 7.5

Index

1 What's New in This Release

What's New in Siebel Product Administration Guide, Siebel 2018

No new features have been added to this guide for this release. This guide has been updated to correct or remove obsolete product and component terms.

NOTE: Siebel 2018 is a continuation of the Siebel 8.1/8.2 release.

What's New in Siebel Product Administration Guide, Siebel Innovation Pack 2017

Developer preview of new servers for Configurator. In Siebel Innovation Pack 2017, the Siebel Server installation includes options for Siebel Rest Proxy Server (based on Apache Tomcat, an open-source Web application server), Enterprise Cache (based on Oracle Coherence, Java-based in-memory data grid software that provides cache management for frequently used data) and the new Constraint Engine. These server installation options are used to integrate Siebel Product Configurator with Oracle Advanced Constraint Technology, providing an alternative to substitute for the current Configurator constraint engine. Oracle provided a developer preview option of this technology in Innovation Pack 2016; this feature continues to be available in Innovation Pack 2017. For more information, see Article ID 2112562.1 on My Oracle Support.

Table 1 lists the changes in this revision of the documentation to support this release of the software.

NOTE: Siebel Innovation Pack 2017 is a continuation of the Siebel 8.1/8.2 release.

Table 1. New Product Features in Siebel Product Administration Guide, Siebel Innovation Pack 2017

Topic	Description
"About the Advanced Constraint Configurator" on page 212	New topic. Information on the Advanced Constraint Configurator has been added.
"Process of Setting Up Advanced Constraint Configurator" on page 212	New topic. Information on setting up the Advanced Constraint Configurator has been added.

2 Overview of Product Administration

This chapter provides an overview of product administration. It includes the following topics:

Product Administration

This guide explains product administration for a number of different types of products. You only have to read the chapters that apply to the types of product your company sells.

Simple Products Without Attributes

This is a product that only comes in one form, such as a book. The customer does not make any decisions about features of the product.

To create simple products without attributes, read Chapter 3, "Basic Product Administration."

Product Bundles

A product bundle is a group of products sold together. It cannot be customized.

For example, you might offer customers vacation packages that include airfare, hotel accommodations for a specific number of days, and specific special events, all for one price.

To create this sort of product, you must read:

- Chapter 3, "Basic Product Administration"
- Chapter 5, "Product Bundles"

Simple Products with Attributes

A product with attributes has features that the customer can choose but does not have components the customer can choose.

For example, a customer buying a t-shirt might be able to choose the shirt's color and its size. The shirt has two attributes, color and size. Each of these attributes has several possible values, such as white, gray, black, and S, M, L, XL.

For information about creating simple products with attributes, see "Roadmap for Creating Simple Products with Attributes" on page 20.

Products with Components

This is a product with components that a customer can select.

For example, a customer buying a computer might have to select a mouse, a floppy disk drive, a monitor, and other components.

For information about creating products with components, see "Roadmap for Creating Products with Components" on page 21.

Product Compatibility

You can define global rules that specify which products and promotions are compatible with each other.

For more information, see Chapter 22, "Product and Promotion Eligibility and Compatibility."

Product Eligibility

You can define global rules that specify which customers are eligible to buy products and promotions.

For more information, see Chapter 22, "Product and Promotion Eligibility and Compatibility."

Product Validation Rules

For special cases where you want to create custom business services to check the compatibility of products, you can use product validation rules.

For more information, see Chapter 23, "Creating Validation Rules for Customizable Products."

Translations

If you are working with any of these types of products and you have to translate the interface into multiple languages, you must also read: Chapter 4, "Multilingual Translations for Product Data."

Roadmap for Creating Simple Products with Attributes

A simple product with attributes has features that the customer can choose but does not have components the customer can choose.

For example, a customer buying a t-shirt might be able to choose the shirt's color and its size. The shirt has two attributes, color and size. Each of these attributes has several possible values, such as white, gray, black, and S, M, L, XL.

To create a simple product with attributes, perform the following tasks:

- **Create the product.** You create this in the same way you create other simple products. See Chapter 3, "Basic Product Administration."
- **Define the attributes of the product.** You must define what attributes the product has and the valid values for each attribute. See Chapter 6, "Products with Attributes"

  For a more advanced method of defining attributes, see Chapter 7, "Product Attributes with Business Component Domains."
- **Decide whether to use the default user interface or create a custom user interface.** For information about the default interface, see "About Default and Custom Siebel Configurator User Interfaces" on page 139.
- **Design the custom user interface.** If you decide to create a custom interface, see Chapter 11, "Creating Custom Siebel Configurator User Interfaces." For more advanced methods of designing a custom interface, see Chapter 12, "Siebel Configurator UI Properties." and Chapter 13, "Siebel Configurator Web Templates."
- **Create constraints for the product with attributes.** For some products with attributes, you create constraints to define which attributes are compatible. For example, a shirt may come in five sizes and ten colors, but some colors may not be available in all sizes.
  - To create simple constraints, see Chapter 14, "Configuration Constraints."
  - For more advanced methods of creating constraints, you can use the same methods used to create advanced constraints for products with components, described in "Roadmap for Creating Products with Components" on page 21.
- **Create scripts for the product with attributes.** Optionally, you can enhance the behavior of Siebel Configurator by writing scripts in the Siebel eScript or the Siebel VB language. When the user selects certain attributes or does things like updating the shopping cart, you can use scripts to check the configuration, verify and adjust pricing, or forward information to other applications. See Chapter 19, "Siebel Configurator Scripts."
- **Create smart part numbers.** If necessary, you can automatically generate a different part number for each combination of attributes that is available. For example, you can have part numbers for size S white shirt, size M white shirt, and so on. This allows you to pass the part numbers to back office applications used for filling orders. See Chapter 8, "Smart Part Numbers for Products with Attributes."
- **Testing Products with Attributes.** After you have designed the product, user interface, and constraints, it is recommended that you test the product to make sure that it works with the products that are available now and in the future. See Chapter 20, "Testing Products and Using Workspace Projects."
- **Releasing Products with Attributes.** After you have tested the product, you can release it to customers. See Chapter 21, "Releasing Products and Other Versioned Objects."

Roadmap for Creating Products with Components

A product with components has components that a customer can select.

For example, a customer buying a computer might have to select a mouse, a floppy disk drive, a monitor, and other components.

Before you create a product with components, perform the following task:

- Decide which rules to use for compatible products. You can define rules that specify which products are compatible by using configuration constraints, compatibility rules, or product validation rules. To decide which to use, see "When to Use Siebel Configurator, Compatibility, Eligibility, and Product Validation Rules" on page 23.

If you decide to use Siebel Configurator, perform the following tasks:

1. **Create the product with components and the component products.** You create these in the same way you create simple products, as described in Chapter 3, "Basic Product Administration." If the product with components or any component products have attributes, create them in the same way you create simple products with attributes, as described in "Roadmap for Creating Simple Products with Attributes" on page 20.
2. **Define the structure of the product with components.** You define the structure of the product with components by specifying which products are its components. See Chapter 9, "Designing Products with Components."
3. For special techniques for defining and managing products with components, see Chapter 10, "Managing Products with Components."
4. **Decide whether to use the default user interface or create a custom user interface.** For information about the default interface, see "About Default and Custom Siebel Configurator User Interfaces" on page 139.
5. **Design the custom user interface.** If you decide to create a custom interface, see Chapter 11, "Creating Custom Siebel Configurator User Interfaces." For more advanced methods of designing a custom interface, see Chapter 12, "Siebel Configurator UI Properties." and Chapter 13, "Siebel Configurator Web Templates."
6. **Create constraints for the product with components.** For most products with components, you must create constraints to define which components are compatible. For example, if the product with components is a computer, you must define constraints to specify which processors are compatible with which operating systems, and so on. To create simple constraints, see Chapter 14, "Configuration Constraints."

   For more advanced methods of creating constraints, you can perform the following tasks:

   - **Designing Links.** Links provide access to other types of information besides products. You can define links to fields in a business component, to the login name of the user, or to the current system date. This lets you write constraints that affect only certain login names, are conditioned on dates, or are conditioned on business component information. See Chapter 15, "Configuration Links."

Designing Resources. Resources keep track of configuration-related amounts in a customizable product. For example, you are designing a customizable product that is a computer. This product has several choices of chassis, each with a different number of card slots. Several of the components in this product are expansion cards that consume these slots. To keep track of the number of slots available you could define a resource called Slots Available. When the user selects a chassis, a constraint associated with the customizable product would add the number of slots in the chassis to

a Slots Available resource. Thus, you can write constraints that monitor slot usage. For more information, see Chapter 16, "Configuration Resources."

- **Modifying Siebel Configurator Constraint Templates.** The Constraints view provides constraint templates that allow you to create a wide variety of configuration constraints. See Chapter 17, "Configuration Constraint Template Reference."
- **Writing Constraints using Siebel Configurator Rule Assembly Language.** Rule Assembly Language (RAL) is for users who are more comfortable working in a programming environment rather than using templates. See Chapter 18, "Siebel Configurator Rule Assembly Language"

7 **Designing Siebel Configurator Scripts.** Optionally, you can enhance the behavior of Siebel Configurator by writing scripts in the Siebel eScript or the Siebel VB language. Scripts allow you to add procedural logic to the configuration process. When the user selects certain items or does things like updating the shopping cart, you can use scripts to check the configuration, verify and adjust pricing, or forward information to other applications. See Chapter 19, "Siebel Configurator Scripts."

8 **Testing Products with Components.** After you have designed the product with components, user interface, and rules, it is recommended that you test the product with components to make sure that it works with the products that are available now and in the future. See Chapter 20, "Testing Products and Using Workspace Projects."

9 **Releasing Products with Components.** After you have tested the product with components, you can release it to customers. See Chapter 21, "Releasing Products and Other Versioned Objects."

10 **Set up cache administration.** You specify how product models will be cached during run time, in order to improve performance. For information about cache administration, see *Siebel Performance Tuning Guide*.

For additional information about products with components, see:

- Chapter 24, "Siebel Configurator Technical Reference." This chapter includes information about a number of features that can be used by developers.
- Chapter 27, "Siebel Configurator Version 6.x, 7.0 and 7.5." If you are upgrading from version 6 to version 7 of Siebel Configurator, read this chapter describing the differences between the products.

When to Use Siebel Configurator, Compatibility, Eligibility, and Product Validation Rules

There are three ways to write rules to specify that products are compatible or incompatible with each other:

- "Configuration Constraints"
- "Compatibility Rules"
- "Product Validation Rules"

In addition, write rules to determine which customers are eligible to buy products using "Eligibility Rules". Use these methods in different cases.

Configuration Constraints

Configuration constraints are used to specify that components of a product with components are not compatible with each other.

For example, a computer is a product with components, and its components that may not all be compatible with each other. A specific model of monitor or keyboard may work only with some CPUs and not with others.

When you define configuration constraints, they only apply within the product with components. If you have many different computers that use the same keyboard, you must write separate configuration constraints for each computer to specify which CPUs that keyboard is compatible with.

Use configuration constraints if the exclude rules and require rules apply to just the component products within a configuration model.

For more information about Siebel Configurator, see "Roadmap for Creating Products with Components" on page 21.

Compatibility Rules

Without configuration, compatibility rules are global. While configuration constraints apply to products only when they are components of a given product with components, compatibility rules apply to products globally.

For example, if you created a rule saying that a given computer keyboard is incompatible with a given CPU, without configuration, the rule would apply whenever that computer and CPU are ordered. It would not apply only to a specific model of computer.

Use compatibility rules without configuration if the exclude or require rules apply across a customers entire asset base, open sales orders and current quote. This is the scope of compatibility rules by default; it is possible to change this scope by configuring the application.

With configuration, it is possible to have compatibility rules apply only to products within the same root product, as configuration constraints do. If you are using only product excludes rules, this approach may be useful, because it avoids the overhead of the Siebel Configurator constraint engine.

For more information about compatibility rules, see Chapter 22, "Product and Promotion Eligibility and Compatibility."

Product Validation Rules

Product validation is most useful when you create your own business services to solve specialized business problems that cannot be addressed using Siebel Configurator.

For more information about product validation rules, see Chapter 23, "Creating Validation Rules for Customizable Products."

Eligibility Rules

Eligibility rules are used to specify which customers are eligible to buy products or promotions.

You must always use eligibility rules for this purpose. You must not use configuration constraints to specify which customers are eligible to buy products.

For more information about eligibility rules, see Chapter 22, "Product and Promotion Eligibility and Compatibility."

About Working with Product Administration

This topic gives you background that you need for working with product administration.

NOTE: It is recommended that you do not use the Product Administration business object layer outside of the Siebel user interface. If you want to create product definitions through scripting, you can use the product import Web services to load product structures, product attributes, classes and relationships, but user interface controls and constraints cannot be loaded. For more information see *Siebel CRM Web Services Reference*. You can also use an external application to build the product structure XML file and automate the import of the XML file into a joint workspace.

Logging On as the Siebel Administrator

The Siebel database server installation script creates a Siebel administrator account that can be used to perform the tasks described in this guide. For more information, see *Siebel Installation Guide* for the operating system you are using and *Siebel System Administration Guide*.

To log on as the Siebel administrator, start the application and log on using the user name and password assigned by your database administrator. Generally, the Siebel administrator connects to the server database.

License Key Requirements

This guide describes basic product management tasks. It also describes how to use Siebel Configurator to create and manage products with components and products with attributes. To use Siebel Configurator, you must have the appropriate license keys installed.

3 Basic Product Administration

This chapter describes the basic product administration tasks common to both simple and customizable products. It includes the following topics:

About the Product Record

Noncustomizable products are called s*imple products*. Products with features that can be chosen at the time of purchase are called *customizable products*. There are two types of customizable products:

- *Products with attributes* have features such as size and color that can be chosen.
- *Products with components* have components that can be chosen.

You enter a product into the database by creating a product record. This record stores important information about the product. The only required field in the product record is the product name. However, it is important to associate the record with a price list and a product line. This allows users to create quotes and to find important information about the product. In addition, when you associate a product with a product class, the product inherits the attributes defined on the class.

Table 2 lists the fields in the product record.

Table 2. Fields in the Product Record

Field	Description
Allocate Below Safety	Click the box to allow allocation below the safe inventory level of this product.
Auto Allocate	Click the box if you are using automatic allocation by the Order Fulfillment engine of a particular product during the fulfillment process.
Auto Substitute	Click the box to allow auto-substitution. Auto-substitution is the automatic use by the Order Fulfillment Engine of a substitute product when the product ordered cannot be found in inventory. The substitute products are set using the Create Substitute form on the Product Field Service Details page.
Check Eligibility	Select this checkbox to make the application check customers' eligibility to buy this product. For more information, see Chapter 22, "Product and Promotion Eligibility and Compatibility."
Compensable	Select this checkbox if sales personnel can receive compensation for selling the product.
Compound Product	Select this checkbox if this is a networking product which uses compound product validation rules.
Customizable	Displays a check mark if this is a customizable product with a work space and at least one version of the product has been released and is available to users.
Description	Enter a brief description of the product.
Division Code (SAP)	Can be used for setting up user access to products but is not recommended. Instead, set up user access by assigning products to categories.
Effective End	The date after which the product is unavailable. This field is for information only. Versioning controls when the product is available.
Effective Start	Enter the date on which the product becomes available. This field is for information only. Versioning controls when the product is available.

Table 2. Fields in the Product Record

Field	Description
Global Product Identifier	Enter a unique product identification string. Use this field to map products from one Siebel installation to another or to a third-party product master. This field is useful when the string in the Part # field is required for local use or is not compatible with third-party product masters. This field is intended for use by integrators needing to move product information between applications.
Equivalent Product	Displays the primary equivalent product. Click the select button in this field to display all equivalent products or to add additional equivalent products.
Field Replaceable	Select this checkbox if this is a field-replaceable unit.
Format	For training products, select a training format such as Instructor led and Web-based.
Image File Name	Select the image file associated with the product. You can also select the image in the Administration - Product screen, Collateral, and then Images. The optimal bitmap dimensions for product images are 4.15 x 4.15 inches, 12.14 x 12.14 cm, or 398 x 398 pixels.
Inclusive Eligibility	Select this checkbox to specify inclusive eligibility. For more information, see “Defining Product and Promotion Eligibility Rules” on page 359.
Integration Id	Enter the back-office application product ID. This field can be used by SAP and Oracle Product Connectors.
Item Size	Enter the numeric product size.
Lead Time	Enter the standard lead time for ordering the product. Measured in weeks. For example, if you enter 2, this means 2 weeks.
Locked Flag	Select this checkbox to lock the product so it can be modified.
Locked By	Displays the user who locked the product.
Model Product	This field is obsolete. It is provided as a reference for upgrade users of Siebel Configurator.
MTBF	Enter the mean time between failures for the product.
MTTR	Enter the mean time to repair the product.
Orderable	Select this checkbox if the product can be ordered. Determines whether a product can be listed as a quote line item on a quote. All components you add to a product with components must be orderable.

Table 2. Fields in the Product Record

Field	Description
Organization	Can be used for setting up user access to products but is not recommended. Instead, set up user access by assigning products to categories. If the default organization is changed accidentally, this change can affect EIM and product migrations, because the application cannot find the default value of the organization while importing the product record.
Parent Product	Select the parent product. This field is for information only. It is not used for creating or managing products with components.
Part #	Enter the part number of the product.
Part Number Method	Select the part number generation methods that can be assigned to a product. This menu is part of the smart part number feature, described in Chapter 8, "Smart Part Numbers for Products with Attributes."
Price Type	Select the price type. Options are: ■ **One-Time.** The customer pays once to buy the product. ■ **Recurring.** The customer pays a fixed recurring fee to use the product. An example is a fixed monthly fee for local telephone service. ■ **Usage.** The customer pays for the product based on usage. An example is the charge for electricity, based on how much you consume.
Primary Vendor	Select the primary vendor for the product. The primary vendor must be specified to associate the product with an opportunity in the Opportunity Product Analysis Chart view.
Product	Enter the name of the product. Products that will be added to the same user access category must not have the same name. The name must be only alphanumeric characters. Special characters such as $ and / are not supported for customizable product scripting or for Siebel Configurator APIs.
Product Class	Select the product class to which you want to assign this product. The product will inherit all the attributes defined on the class or that are inherited by the class. This is the field in the Workspace version. For the information to take effect, you must click Release to release the product.
Product Level	Enter the numeric product level in the product hierarchy. This field is for record keeping only and is not used to create or manage the product class system.
Product Line	Select the desired product line for the product.

Table 2. Fields in the Product Record

Field	Description
Project Resource	Select this checkbox if the product is a service for a project. This determines if the product is going to be available in the rate list.
Qty	Enter the number of items in the unit of measure. For example, if the unit of measure is a case, Qty would be the number of items in the case, such as 24.
Return if Defective	Select this checkbox to indicate that a defective product must be returned by the customer when a replacement is shipped. Deselect the checkbox if customers must not return defective products.
Revision	Select the revision level of the product as it goes through revisions.
Sales Product	Select this checkbox if the product is a sales product. This determines whether the product is displayed in the Product picklist for Opportunities.
Serialized	Select this checkbox if movement of the product (a transaction) requires an asset number or its corresponding serial number. The default is no check mark or X (not serialized). **NOTE:** If the Serialized field is selected, you must enter the Asset # field of the Quote item, Order item or Asset.
Service Product	Select this checkbox if the product is a service. Only products designated as service products will display when you click the Service button on a quote. Special pricing rules apply to service products. For more information, see *Siebel Pricing Administration Guide*.
Ship Carrier	Select the name of the shipping carrier for this product.
Ship Method	Select the shipping mode: air ground, and so on.
Status	Select the status of the product: prototype, alpha, beta, and so on.
Structure Type	Select the type of structure the product has. Options are: ■ **None.** Simple product. ■ **Bundle.** Bundle product. ■ **Customizable.** Customizable product. Structure Type controls whether the Customize button appears in the product selection, quote, and order user interface. Structure Type does not control how the product appears in the Siebel Configurator User Interface. A product appear as a customizable product in Siebel Configurator as long as it has either attributes or components that the user can select, regardless of Structure Type.
Targeted Country	Select the country where you want to sell this product.

Table 2. Fields in the Product Record

Field	Description
Targeted Industry	Select the industry you want to target with this product.
Targeted Max Age	Enter the maximum age of buyers for this product.
Targeted Min Age	Enter the minimum age of buyers for this product.
Targeted Postal Code	Enter the postal code where you want to target sales of this product.
Tax Subcomponent Flag	Select this checkbox to compute the tax on a bundle by adding up the tax on its components. Useful when the tax rate or computation method is not the same for all the components in a bundle.
Taxable	Select this checkbox if the product is taxable.
Thumbnail Image File Name	Select the thumbnail image file associated with the product. You can also select the thumbnail image in Product Administration, then Product Images.
Tool	Click the box if this product is a tool, such as one used by field service engineers.
Track as Asset	Select this checkbox if, when the product is purchased, you want to track it as a customer asset to allows you to create quotes and orders based on the asset. For more information, see the topic about asset-based ordering in *Siebel Order Management Guide*.
Type	Select the product type: product, service, or training. You must select a Type if users will be using the Spread Discount feature in Quotes. If you create custom values in this list, you must configure the product using Oracle's Web Tools to make the Spread Discount feature work. For more information about Spread Discount, see *Siebel Order Management Guide*.
Unit of Measure	Select the unit of measure by which the product is sold, for example, Each.
Vendor Part #	Enter the vendor's part number for this product.
Vendor Site	Displays the primary vendor's location. This field is filled automatically when you select a vendor.

Process of Creating Simple Products

This process covers the essential tasks that you must perform to create a simple product and make it visible to users. Other tasks for creating simple products are covered in the rest of this chapter.

To create a simple product, perform the following tasks:

- "Creating a Product Record" on page 33
- "Associating a Product with Price Lists" on page 33
- "Setting Up User Access To a Product" on page 33

■ "Releasing a Simple Product" on page 34

Creating a Product Record

You enter products into the Siebel application by creating product records. The product record contains the product name and important information about the product, such as its product line name or part number.

You add a new product record by clicking the New button. This creates the new product without releasing it, and locks the workspace.

This task is a step in "Process of Creating Simple Products" on page 32.

To create a product record

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, add a new record and complete the necessary fields, described in "About the Product Record" on page 27.

Associating a Product with Price Lists

Products are not visible to customers unless they are associated with price lists that are assigned to the customers.

For more information on creating price lists and assigning them to customers, see *Siebel Pricing Administration Guide*.

This task is a step in "Process of Creating Simple Products" on page 32.

To associate a product with a price list

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product.
3. Click the Pricing view tab.
4. In the Price Lists list, add a new record and select the desired price list.
5. Complete the remaining fields as needed.

   For more information about these fields, see *Siebel Pricing Administration Guide*.

Setting Up User Access To a Product

You must set up user access to allow the user to select a product for a quote or to see the product in a catalog.

The catalog administrator creates product catalogs, which contain product categories. The catalog administrator sets up access controls by assigning access groups to the catalog and to the categories. For information about creating catalogs and categories and giving users visibility to them, see *Siebel Order Management Guide*.

The product administrator assigns products to catalogs and categories. You can assign a product to more than one category, and thus more than one catalog.

Until you assign a product to at least one category, the product does not display in the following places:

- **On eSales Web pages.** When customers buy your products through the Web, they cannot see the product.
- **While browsing catalogs.** When salespeople click the Browse Catalog button to view products in catalogs, they cannot see the product.

NOTE: For products with components, you must give users access to the product with components and all its components. To accomplish this, first assign the product with components and its components to the same product category or to categories that have the same access groups. Then assign users who will configure the product to these access groups. If the users in the access groups differ across components, these users will not be able to configure the product with components correctly.

The recommended method for assigning users to access groups is to assign the users to organizations and then assign the organizations to the access groups.

This task is a step in "Process of Creating Simple Products" on page 32.

To set up user access

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product.
3. Click the Category view tab.
4. Add a new record to the Category list.

   A dialog box appears that lists all the currently defined categories.
5. Select a category from the dialog box.
6. Repeat Step 4 and Step 5 to add all the categories needed to give users visibility to this product.

Releasing a Simple Product

Though simple products do not have a Versions list, they are versioned objects. A new version was created when you created the product, and you must release this version to make the product visible to users.

For more information about versioned objects, see Chapter 21, "Releasing Products and Other Versioned Objects."

This task is a step in "Process of Creating Simple Products" on page 32.

To release a simple product

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the simple product.
3. Click Release.

Setting Up Products with Recurring Prices

Some products involve recurring prices. For example, customers pay a monthly fee for telephone service. For more information, see the topic about multiple price types in *Siebel Pricing Administration Guide*.

To set up a product with recurring prices

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product.
3. In the More Info form, confirm that the fields are filled out appropriately, as described in the following table:

Field	Comments
Price Type	Confirm that this is set to Recurring.
Unit of Measure	Confirm that this is set to an appropriate value for a recurring price, such as Per Month.

Creating Product Lines

Product lines are used to group your products.

For example, if you sell clothing, men's shirts may be one product line, women's shirts may be a second product line, and so on.

NOTE: You can add products to a product line by selecting them when you add the Product Line record, as described in this procedure. When you create a new product that is part of the product line, you can add it to the product line by selecting it in the Product Line field of the Product record.

To create a product line name

1. Navigate to the Administration - Product screen, then the Product Lines view.

2 In the Product Lines list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Product Line	Enter a name for the product line.
Product Line Manager	Optionally, select product line managers and other key personnel associated with the product line.
Products	Select all the products in this product line.

Defining Product Features

Products frequently share common features, such as size or data transfer rate. You can create a list of these product features and assign features to products.

Product features are different from product attributes:

- A product feature describes important characteristics of a product, particularly those that differentiate the product. For example, you sell a type of office chair that has aluminum construction. Your competitors sell the same office chair with steel construction. Aluminum construction is an important feature of the office chair because it differentiates the chair from your competitors. It is also a static feature and cannot be chosen by the customer. All of your customers who purchase this office chair get aluminum construction.
- A product attribute is a characteristic of a product that the customer can choose when purchasing the product. For example, the office chair fabric comes in one of three colors. Color is an attribute of the office chair because the user can choose the color at the time of purchase.

To define product features

- create the product features.
- associate the product features with products.

To create product features

1 Navigate to the Administration - Product screen, then the Product Features view.

2 In the Product Features list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Feature	Enter the name of the product feature.
Product Line	Optionally, select a product line that is associated with this product feature.

The application adds these features to the Features picklist, so you can assign them to individual products.

To assign a key feature to a product

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the product to which you want to assign a key feature.
3. Click the Collateral view tab.
4. In the Collateral link bar, click Key Features.
5. In the Key Features list, add a new record and select the feature in the Features field.
6. Repeat Step 5 to add additional key features for the product.

Defining Related Products

You can define several types of relationships between products. This causes the related products to appear together in other parts of the Siebel application.

For example, if you define a substitute product in the Related Products view, the substitute product displays in the Product Service Details view. If you define a substitute product in the Product Service Details view, it displays automatically as a substitute product in the Related Products view.

You can define the following types of relationships:

- Bundled
- Component
- Cross-Promoted
- Integrated
- Recommended Service
- Service
- Substitute

To define related products

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the product with which you want to associate related products.
3. Click the Recommendations view tab.
4. In the Recommendations view link bar, click Related Products.
5. In the Related Products list, add a new record.

   The Add Internal Products dialog box appears.
6. Select the desired product.

   The product appears in the Related Products list.
7. To change the relationship of the related product, click in the Relation field and choose the desired relationship from the drop-down menu.

Defining Equivalent Products

For each product you define, you can designate one or more other products as equivalent products. You can then display these products and compare their product features. You can also assign a ranking to the equivalent products that reflects their degree of equivalence.

Equivalent products differ from substitute products in that they do not automatically display in the Field Service Details view.

You can designate one of the equivalent products as the primary equivalent product. The equivalent primary product is the one displayed in the Equivalent Products field in the product definition and other places where the display allows only one equivalent product to be shown.

To designate equivalent products

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product.
3. In the More Info form, click the show more button to expand the form.

   Expanding the form displays the Equivalent Product field.
4. Click the select button in the Equivalent Product field.
   a. Use the Equivalent Products dialog box to add equivalent products.
   b. In the dialog box, select the Primary field for one product that you are adding to designate it as the primary equivalent product.
   c. Click OK to exit the dialog box.

   The primary equivalent product appears in the Equivalent Product field in the product record.

Comparing Features of Equivalent Products

You compare equivalent products by displaying all the equivalent products for a product and then selecting which features you want to use for the comparison. You can then rank the equivalent products.

To compare features of equivalent products

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product.
3. Click the Collateral view tab.
4. In the Collateral view link bar, click Key Features.

   The Product Comparison list appears beside the Key Features list. Equivalent products are displayed in the columns of the Product Comparison list.

5 In the Product Comparison list, add a new record.

A dialog appears that contains all the product feature definitions.

6 Select the desired product from the dialog box.

The feature is added to the Product Comparison list.

7 Repeat these steps until all the desired features have been added.

8 Assign a ranking to the equivalent products, if desired.

A rank of 1 means a product has the highest degree of equivalence relative to the other equivalent products.

Creating Product Entitlements

Entitlements refer to the services that come with a product. They are created on the Product Entitlements page under Product Administration.

When you create a product entitlement, you can designate the entitlement as applicable to “Agree Line Item Products” and/or “Entitlement Template Products.” These are for Field Service use. For more information, see *Siebel Field Service Guide*.

Entitlement templates are used for different purposes in the Administration - Product screen and in the Administration - Service screen:

- **Administration - Product screen.** If you associate product with an entitlement template in the Administration - Product screen, as described in the following procedure, any customer who buys that product will automatically have those entitlements.
- **Administration - Service screen.** If you associate product with an entitlement template in the Service Administration, Entitlement Templates, and then the Products view, you indicate that this product is covered by the entitlement. When this entitlement template is used in a contract, the contract will automatically cover all the products listed under the entitlement templates.

To create product entitlements

1 Navigate to the Administration - Product screen, then the Products view.

2 In the Products list, select the product for which to create entitlements.

3 Click the Service Information view tab.

4 In the Service Information view link bar, click Entitlements.

5 In the Entitlements list, add a new record.

6 Click the select button in the Name field and select an Entitlement template from the Entitlement Templates dialog box.

The entitlement template record is added to the Product Entitlements list.

7 Click in the Agree Line Item or Entitlement Template Products field to set these features.

A check mark appears to indicate these features are set.

Associating Literature with Products

You associate literature with products so salespeople can use this literature to sell the products. Product literature is associated with the product as an attachment, so it can be used for such things as product brochures, competitive analyses, and image files.

NOTE: When you choose literature to associate with a product, only literature of the type Sales Tool is displayed. When you create literature to be associated with products in this way, be sure to choose Sales Tool in the Type field. For more information about creating literature, see *Siebel Applications Administration Guide*.

To associate literature with a product

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product.
3. Click the Collateral view tab.
4. In the Collateral view link bar, click Literature.
5. In the Literature list, add a New Record.

   The Add Literature dialog box appears.
6. Select the desired literature items.

Associating Product News with Products

Product news is information about a product that is displayed in Self-Service and eSales as inline text associated with the product.

Product news is not the same as product literature, which is covered in "Associating Literature with Products" on page 40.

To add a news item to a product

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select a product to which you want to add a news item.
3. Click the Collateral view tab.
4. In the Collateral view link bar, click News.
5. In the News list, add a new record.

   The Pick Product News dialog box appears. To read the first few lines of a news item in the dialog box, place your cursor over it.
6. Select the desired news item.

   The news item appears under Product News with its title under the Solution field and the solution type set to Product News.

7 Edit the record as needed by clicking the desired field.

Uploading a New Image in the Application

You can upload new images in the application.

To upload a New Image in the Application

1 Navigate to the Administration - Document screen, then Literature.

2 Click New to upload a new image.

Associating Images with Products

You can associate both a thumbnail image and a regular image with a product.

To associate images with a product

1 Navigate to the Administration - Product screen, then the Products view.

2 Click the Collateral view tab.

3 In the Collateral view link bar, click Images.

4 In the Images form, in the Image File Name field, select an image.

5 In the Images form, in the Thumbnail Image File Name field, select an image.

Creating Product Field Service Details and Measurements

You provide information about how to replace a defective part with substitute parts in the Administration - Product screen, then the Product Service Details view. Most field service information is entered when creating products in the Products view, but Inventory Options and Substitute Products are managed in the Product Field Service Details view.

The Administration - Product screen, then the Measurements view is used to define which measurements field service personnel must make and what the parameters of those measurements must be.

For more information about both of these, see *Siebel Field Service Guide*.

Exporting and Importing Products

You can import and export products using Workspace Projects view. For more information, see "Migrating Products Among Environments" on page 351.

About Managing Product Records

You can manage product records in the following ways:

- "Editing Product Records" on page 42
- "Copying Product Records" on page 42
- "Deleting Product Records" on page 43
- "Exporting Product Records for Display" on page 43

Editing Product Records

You can change the content of any of the fields in a product record. Changing the class to which a product is assigned can change the attributes the product inherits. If the product's attributes change, you must revise all products with components in which the product is component. Verify that no configuration rules or scripts refer to attributes the product no longer has.

CAUTION: If you change the name of the product, you must revise all products with components in which this product is a component. Also revise the configuration rules, UI design, and scripts that refer to the product.

Copying Product Records

When you copy a product record, all parts of the product definition are included in the copy.

If you copy a customizable product record, the copy includes all the relationships, links, resources, scripts, rules, and user interface of the product version in the workspace.

Use the Copy feature to create product templates. For example, your product line has a two-tiered structure. The first tier contains a half-dozen products that have a similar basic structure. The second tier contains products based on the structure of the products in the first tier.

You could create the first tier by copying a template product with components 6 times. You would then modify each of the copies to form the first tier. These then become the templates you would use to create the second tier.

To copy a product record

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the product you want to copy.

3 From the menu, choose Copy Record.

A new record appears.

4 Enter a name for the copy in the Product Field.

5 Revise other fields, such as Part # as desired.

Deleting Product Records

You cannot delete a product record. If you no longer want to use the product, you can deactivate all versions rather than deleting the product.

If you have a large number of inactive versions for a given product, you can delete the versions using the CleanupSingleObject method of the ISS Authoring Import Export Service. For more information, see *Siebel Order Management Guide*.

To deactivate a product

1 Navigate to the Administration - Product screen, and then the Products view.

2 In the Products list, select and lock the desired product.

3 In the Versions view, deselect the Active checkbox for the Work Space version.

4 Release the Work Space version.

Exporting Product Records for Display

You can export product records in several formats for display.

For example, you can download files in comma-separated format for display in Microsoft Excel. The supported formats are as follows:

- Tab delimited file
- Comma separated file (csv format for use with spreadsheets like Excel)
- HTML file
- A file with delimiters you specify

You can request all the rows in the current query or only the highlighted rows. You can request all columns or only the currently visible columns. Currently visible columns are those you have selected for display in the Columns Displayed form.

When you export a product with components or bundle for display, only the root-level product record is exported. The structure of the product with components or bundle is not exported.

NOTE: This procedure exports only product records for use in other display mediums such as spreadsheets. This procedure does not export the structure of a product or any other information contained in records related to the product record. To export product structures and other information in XML format for use by other applications, see "Exporting and Importing Products" on page 42.

To export product records for display

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the products you want to export.
3. Verify that the columns displayed are those you want to export.

   To add or subtract columns, from the Products menu, choose Columns Displayed.
4. To export the product records for display, from the Products menu, choose "Export..."

   Do not click Export Product. This will export the product information in XML format for use by other applications.

   The File Download dialog box appears.
5. Follow the instructions in the File Download dialog box to save the file.

Creating a Product List Report

You can obtain a report that lists all the products in the product table. For each product, the report shows the following information:

- Product name
- Part number
- Description
- Unit of measure
- Vendor
- Product line
- Effective start date
- Effective end date

The product list displays in the Siebel Report Viewer. You can print the report or create an email attachment.

TIP: The on-screen display of the report typically lists more products on each page than the Products list. Use the report to scan through the product table.

To create a product list report

1. Navigate to the Administration - Product screen, then the Products view.
2. Click the Reports button at the start of the screen, and from the Reports menu, select Admin Product List.
3. Use the dialog box to run the report.

   The Siebel Report Viewer appears and displays the Admin Product List report.
4. Print the report or create an email attachment as desired.

Clearing the Siebel Configurator Cache to Improve Performance

The best practice is to force a full resynchronization of the Siebel Configurator cache periodically. For example, you might resynchronize the cache once a month, when there is a low load on the Siebel application. How often you should actually resynchronize the cache depends on how rapidly changes to versioned objects accumulate over time in your implementation.

Two special tables, S_VOD_CASHESYNC and S_VOD_CACHEREQ are used to drive the synchronization process and are queried frequently. Records in those tables are hold temporary information that facilitate cache synchronization. This information can accumulate over time, causing performance degradation. You notice this degradation as queries become slower.

To force resynchronization, select the menu option that invokes the ResetSKC method to clear the full Siebel Configurator cache for Products, Product Classes and Attribute Definitions.

To clear the Siebel Configurator cache to speed queries

1. Navigate to the Administration - Products screen, then the Cache Administration view.
2. From the applet menu, select Refresh Cache.

4 Multilingual Translations for Product Data

You can specify language translations for product-related data the user sees when creating a quote or purchasing a product from an eSales Web site. This chapter describes what product data can be translated and how to specify the translations. It includes the following topics:

About Product Data Translation

You can specify language translations for the following data:

- Product description
- Product class display name
- Attribute display name
- Attribute definition name
- Attribute list of values

In addition, for products with components, you can translate the following data:

- Configuration rule explanation
- Relationship name
- UI group name
- UI property value

The process for translating each of the types of product data is the same. The Product Administrator selects the desired item, selects a language, and then enters the translation for the item. This creates a record containing the translation. The Product Administrator can create multiple translation records for an item.

When users log in to either Quotes or to an eSales Web page and specifies a language, they see the item translations for that language entered by the Product Administrator.

In some cases, the lists that display items that can be translated include a field called Translate. This field is unrelated to setting up data for multilingual translation and must be ignored.

Translating the Product Description

Use these procedures to translate the product description.

To translate the product description

1. Navigate to the Administration - Product screen, then the Products view.
2. Select a product whose description you want to translate.
3. Click the Translations view tab.
4. Add a new record to the Translations list and complete the necessary fields, described in the following table.

Field	Comments
Language	Displays the name of the language after you select the code.
Code	Select a language code.
Description	Enter the translation of the description.

To translate the product description of customizable products

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Product Definitions list, select a customizable product whose description you want to translate.
3. In the Versions list, click the name of the Work Space version.
4. Click the Display Names view tab.
5. Add a new record to the Display Names list and complete the necessary fields, described in the following table.

Field	Comments
Display Name	Enter the translation of the name.
Language	Select a language code.

Translating Product Class Display Names

You can enter translations of the names of product classes, so they are displayed in the language of the end user. To translate a class display name

1. Navigate to the Administration - Product screen, then the Product Classes view.
2. Select and lock product class whose attributes you want to translate.
3. In the Versions list, click the Work Space version.
4. Click the Display Name view tab.
5. Add a new record to the Display Name list and complete the necessary fields, described in the following table.

Field	Comments
Display Name	Enter the translation of the display name.
Language	Select a language code.

6. Repeat Step 5 to create additional language translations for the class display name.

Translating Attribute Names

You can translate the name of an attribute, so it is displayed to end users in their own language, in the following ways:

- You can translate the attribute name at the Product Classes level, so all products in the class inherit the translation of the attribute name.
- You can translate the attribute name at the Product level, so the translation applies only to that product.

To translate an attribute display name at the Product Classes level

1. Navigate to Administration - Product screen, then the Product Classes view.
2. Select and lock the product class where the attributes are defined.
3. Click the Class Attributes view tab.
4. In the Versions list, click the Work Space version.
5. Click the Attributes view tab.
6. From the Attributes menu, select Translations.

7 Add new records in the Translations dialog box and complete the necessary fields, described in the following table.

Field	Comments
Language	Select a language code.
Name	Enter the translation of the name.

To translate an attribute display name at the Product level

1 Navigate to Administration - Product screen, then the Product Definitions view.

2 In the Products list, select the product whose attributes you want to translate.

3 In the Versions list, click the name of the Work Space version.

4 Click the Attributes view tab.

5 From the Attributes menu, select Translations.

6 Add new records in the Translations dialog box and complete the necessary fields, described in the following table.

Field	Comments
Language	Select a language code.
Name	Enter the translation of the name.

Translating Attribute Definition Names

You can translate the name of an attribute definition, so it is displayed to end users in their own language.

To translate an attribute definition

1 Navigate to Administration - Product screen, then the Attribute Definitions view.

2 In the Attribute Definitions list, select and lock the desired attribute definition.

3 In the Versions list, click the name of the Workspace version.

4 In the Attribute Values list, select an attribute value.

5 In the Attribute Values Display Names list, add the new records, and complete the necessary fields, as described in the following table.

Field	Comments
Display Name	Enter the translation of the attribute definition name.
Language	Select a language code.

6 Continue to enter translations for all the values of this attribute.

Translating Attribute Values

For attributes with a list of values domain, you can translate the attribute values. For example, you have a list of values named Color with the values red, blue, and green, and you want to translate these values into French, Spanish, and other languages.

For additional information on creating and managing multilingual lists of value (MLOVs), see *Siebel Global Deployment Guide*.

To translate an attribute list of values

1 Navigate to the Administration Product screen, then the Attribute Definitions view.

2 In the Attribute Definitions list, select and lock the attribute whose values you want to translate.

3 In the Versions list, click the Work Space version.

4 Click the Details view tab.

5 In the Attribute Values list, select the attribute value that you want to translate.

6 In the Attribute Value Display Names list, add a new record for each language that you want to translate this value into, and complete the necessary fields, described in the following table.

Field	Comments
Display Name	Enter the translation of the value.
Language	Select a language code.

7 Repeat Step 5 and Step 6 to translate all the values in the Attribute Values list.

Translating Configuration Rule Explanations

Use this procedure to translate configuration rule explanations for a customizable product.

To translate a configuration rule explanation

1 Navigate to the Administration - Product, then the Product Definitions view.

2 In the Products list, select and lock the desired customizable product.

3 In the Versions list, click the Work Space version.

4 Click the Constraints view tab.

5 In the Constraints List, select the constraint containing the explanation you want to translate.

6 From the Constraints List menu, choose Translate Constraint Description.

A dialog box appears that displays the rule explanation translations you have already created.

7 In the dialog box, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Language	Displays the name of the language after you select the code.
Code	Select a language code.
Description	Enter the translation of the description.

Translating Relationship Names

You can translate relationship names for a product with components the following ways:

- You can translate the relationship name at the Product Classes level, so all products in the class inherit the translation.
- You can translate the relationship name at the Product level, so the translation applies only to that product.

To translate a relationship name at the Product Class level

1 Navigate to the Administration - Product screen, then the Product Classes view.

2 Select and lock the desired Product Class.

3 In the Versions list, click the name of the Work Space version.

4 In the Structure list, select the relationship whose name you want to translate.

5 From the Structure list menu, choose Translate Relationship.

A dialog box appears that displays the relationship name translations you have already created.

6 In the dialog box, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Language	Select a language code.
Relationship Name	Enter the translation of the relationship name for that language.

To translate a relationship name at the Product level

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Products list, select and lock the desired product with components.

3 In the Versions list, click the Work Space version.

4 Select the relationship whose name you want to translate.

5 From the Structure list menu, choose Translate Relationship.

A dialog box appears that displays the relationship name translations you have already created.

6 In the dialog box, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Language	Displays the name of the language after you select the code.
Code	Select a language code.
Description	Enter the translation of the description.

Translating UI Group Names

Use this procedure to translate group names that display in customizable product selection pages.

To translate a UI group name

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Products list, select and lock the desired customizable product.

3 In the Versions list, click the Work Space version.

4 Click the User Interface view tab.

5 In the User Interface view, click the Name of the UI group whose name you want to translate to drill down on it.

6 From the Group List menu, choose Translate Groups.

The Group Name Translations dialog box appears.

7 In the dialog box, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Language	Select a language.
Name	Enter the translation of the group name.

8 Repeat Step 7 to create additional translations for this UI group name.

Translating UI Property Values

Use this procedure to translate the value of a UI Property. The property type must be type String.

To translate a UI property value

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired customizable product.
3. In the Versions list, click the Work Space version.
4. Click the Properties view tab.
5. Select the UI property you want to translate.
6. From the Customizable Product menu, choose Translate UI Property.

   A dialog box appears that displays the relationship name translations you have already created.
7. In the dialog box, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Language	Displays the name of the language after you select the code.
Code	Select a language code.
Description	Enter the translation of the description.

5 Product Bundles

This chapter explains how to create product bundles. A product bundle is a group of products that are sold together for a specified price. It includes the following topics:

About Product Bundles

A product bundle is a group of products sold as a package. If you create a product bundle, the user cannot change the items in the bundle or their quantity. If you want the user to be able to select the items, you must use a product with components instead of a product bundle.

A product bundle is itself a product and has a product record. It can also have a part number. You price bundles by assigning them a list price. You cannot use Pricer to create roll-up pricing based on components or on attributes of components in a bundle. You also cannot use attribute-based pricing to set the price of a bundle based on the attributes of the bundle as a whole.

When you create a bundle, it is added to the product master. This means you can add the bundle to any quote or order. Product packages that you create in a quote or order are bundles that are specific to that quote or order. They are not added to the product master.

Creating Simple Product Bundles

A simple product bundle is a group of products offered as package. The user cannot change the items in the bundle or their quantity.

To create a simple product bundle, you first create a product record for the bundle and select Bundle in the Structure Type field. Then you add products to the bundle. After creating the bundle, see *Siebel Pricing Administration Guide* to set up pricing.

Observe the following guidelines and restrictions when creating a simple product bundle:

- The quantity of a product in a bundle can be greater than one. When creating a quote or purchasing the bundle, users cannot change the quantity of a product in the bundle. The user cannot change which items are in the bundle.
- When users add bundles to quotes and orders, the products in the bundle display as line items beneath the bundle's product name.

- You can add a product bundle to another product bundle.
- You can add a product with components to a bundle.
- When you add a bundle to a product with components, the user can change the quantity of the bundle product during a configuration session.
- You can convert a bundle to a product with components and you can convert a product with components to a bundle.

To create a product bundle

1. Navigate to the Administration - Product screen, then the Products view.
2. Create a new product record, enter the name of the bundle in the Product field, and select Bundle in the Structure Type field.

   Complete any other needed fields for the product bundle, as you would for a simple product.
3. Click the Bundle Product view tab.
4. In the Product Bundle list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Product	Select a product that is in the bundle.
Description	Enter a brief description of the bundle. This does not display to users.
Quantity	Enter the quantity of the product you want to include in the bundle.
Sequence	Enter the order in which products in the bundle display in quotes and orders.
Forecastable	Select this field to add the product to product forecasts when the bundle is included in a quote and the user updates the related opportunity.

5. Repeat Step 4 for each product you want to add to the bundle.
6. Create the pricing for the bundle product, as described in *Siebel Pricing Administration Guide*.
7. Click Done.

   This releases the bundle for use by customers. A check mark displays in the Bundle check box in the product record form.

Modifying Simple Product Bundles

You can modify a simple product bundle by changing the items in the bundle or by changing the quantity of items. Modifying a product bundle releases a new version of the bundle.

To modify a product bundle

1. Navigate to the Administration - Product screen, then the Products view.
2. Select and lock the record for the desired bundle.
3. Click the Bundle Product view tab.
4. Click Modify to edit, add, or delete products from the bundle.
5. When finished, click Done.

   This releases a new version of the bundle for use by customers.

Deleting Simple Product Bundles

You cannot delete the product record for a product bundle. However, you can make the product bundle unavailable for use.

To make a product bundle unavailable

1. Modify the bundle to remove all its products.
2. In the bundle's product record, deselect the Sales Product check box.

   This removes the product bundle from the product picklist.
3. Remove the product bundle from all price lists.
4. Delete any pricing rules that refer to the product bundle.
5. Remove the product bundle from all product with components relationships, and configuration rules. Validate the products with components and release a new version.

Controlling How Bundle Components are Forecast

When you add a product to a bundle, you can put a check mark in the Forecastable field. This adds the product to forecasts when the bundle is included in a quote and the user updates the related opportunity.

To prevent bundle products from being added to product forecasts, do not put a check mark in the Forecastable field in Bundle Administration.

A Forecastable check box is also available in the Quotes screen, then the Line Items view. This allows you to add or remove a bundle and its products from product forecasts within individual quotes.

6 Products with Attributes

This chapter describes how to create product classes and product class hierarchies. Product classes provide a central location for defining product attributes. Products inherit the attributes of the product classes to which they belong. This chapter includes the following topics:

Component-Based Versus Attribute-Based Pricing

Users to select the features of two types of customizable products:

- **Products with components.** Customers can choose the product's components. For example, when customers buy a computer, they can choose the monitor, keyboard, and mouse they want.
- **Attribute-based products.** Customers can choose the product's attributes. For example, when customers buy a shirt, they can choose its color.

In most cases, it is clear which one of these two a product is. However, in some cases, the product administrator must decide whether to let users select options as product components or as attributes. For example, if a PC monitor comes in three sizes, 14-inch, 17-inch, and 21-inch, then those monitors could be offered as:

- Three separate products with three different prices, which are used as three component products within a customizable PC product.
- One monitor product with a Screen Size attribute that includes three values (14, 17, and 21), which is used as one component product within a customizable PC Product.

You must decide how to set up products with components, in a way that will be effective as the product hierarchy and its circumstances change.

About Product Attributes

Product attributes are characteristics of a product that a customer can choose. For example, you sell a product in three colors. As part of creating this product, you would define an attribute called Color and assign it the three colors. As part of purchasing the product, customers would choose one of the colors.

A product attribute has two parts: the name of the attribute and the possible values of the attribute. For example, you could define an attribute with the name Color and the values red, green, or blue. The allowable values for an attribute are called the attribute domain. In a configuration session, the user can select only one value for an attribute.

Components of a product are not attributes. For example, you sell a desktop computer. Customers can select one of several types of CD-ROMs when configuring this product. Having a CD-ROM is a characteristic of this product, but the CD-ROMs are components, not attributes.

Product attributes and product features are similar concepts. They both describe characteristics of the product that are of interest to customers. However, feature definitions do not create configurability. For example, you could define a feature: “Comes in three colors, red, green, and blue.” This feature definition can be displayed to the user as a message only. It does not create the mechanism for choosing the color. To create that, you must define a product attribute and assign it the values red, green, and blue.

You can define attributes directly in the administration interface. You do not need to create database table extensions or new field definitions in Oracle’s Web Tools.

Attributes are implemented in a way that allows users to select the desired attribute value when they configure the product. For example, when a user creates a quote, the Color attribute displays in the interface, and the user can select the desired value.

About Product Classes

Product classes provide a way to organize and administer product attributes. When you assign a product to a product class, it automatically inherits all the attributes defined for that product class. Product classes let you define what attributes are maintained for products, assign those attributes to the products, and maintain those attributes in a consistent fashion.

When you define an attribute for a product class, you specify both the attribute name and the range of values that the attribute can have. This range of values is called the attribute domain. For example, for a product class called blanket, you define an attribute called color and define its domain to be green, red, and blue. Every blanket you assign to this product class inherits the attribute color and its possible values.

Subclasses are product classes that have a parent product class. Subclasses have the following characteristics:

- Subclasses can be nested as deeply as needed. This forms the product class hierarchy.
- Subclasses inherit the attributes of their parent product class. As you nest downward, each subclass inherits the entire set of attributes from the product classes above it.
- You can modify the definitions of inherited attributes. If you do so, this breaks inheritance from the parent product class. Changes to attribute definitions in the parent product class are not inherited by modified attributes in subclasses.
- You can define additional attributes for the subclass, beyond the attributes of the parent product class.

You can define attributes at the product class or subclass level. You cannot define an attribute at the product level. At the product level, users can only select the attribute's value.

About the Product Class Hierarchy

The product class hierarchy allows you to organize and manage product attributes. It is separate from the mechanisms you use to organize products themselves, such as product lines and product categories.

For example, you have the product class hierarchy in Figure 1. The product class called Class has two attributes defined on it, Attribute 1 and Attribute 2. Class also has a subclass called Subclass. Subclass has Attribute 3 defined on it and contains one product, called Product C.

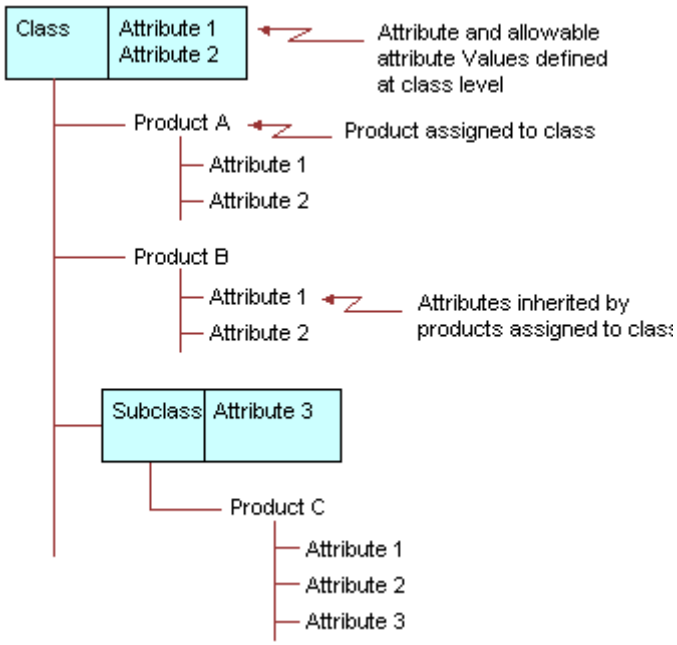

Figure 1. Class Hierarchy

Subclass inherits Attribute 1 and Attribute 2. It also has an attribute definition of its own, Attribute 3. Product C, assigned to Subclass, inherits all three attribute definitions.

NOTE: When you define a product with components, you define named parts called *relationships*. Then you add the contents of product classes to them. Adding a small number of products to a relationship from a large product class requires that the entire product class be searched each time the product with components is instantiated. This can adversely affect performance. Consider defining the product class system to avoid this.

In the Administration - Product screen, then the Product Classes view, you can create product classes, organize them into hierarchies, and define attributes for them. By clicking the Structure view tab, you can view the Class Explorer, which gives you an overview of the entire product class hierarchy system.

About Attribute Domains

When you define an attribute, you must define the domain of possible values for the attribute.

About Defining Attribute Domains

There are the following methods for defining the domain:

- **List of values.** You can list the specific values the attribute can have. When users configure a product, they select one of the values from a drop-down menu. For example, the attribute Color could have the list of values red, green, or blue.

  A special case of a list of values domain is a list of values that contains only one value. This is useful for creating attributes that you use for managing resources. For example, you could create an attribute called slots-consumed for a product class of computer expansion cards. Typically, each card requires one expansion slot. You would create a list of values containing only the number 1, and would set 1 as the default value. You could then write rules that subtract the value of this attribute from a resource called slots-available each time the user picks an expansion card.

  Parametric search can be used to search for attribute values.

  Attribute-based pricing can only use attribute values that have been defined as elements in a list of values (LOV). Attribute-based pricing requires the discrete values that appear in an LOV.

- **Free form.** This domain allows free form user input. During runtime, it provides a blank field where the user can make any desired entry.

  Parametric search cannot be used to search for attribute values.

- **Business Component (Buscomp) field.** This domain is defined by a field in a business component. For example, you can define an attribute called Account and associate it with the Name field in the Account business component. When users configure a product, they see an attribute called Account. They can then open a picklist and select the desired account. This domain type can be used only for products that are configured in Siebel Configurator selection pages.

  Parametric search cannot be used to search for attribute values.

This chapter covers the list of values and free form domain types. For information about the Business Component field type, see Chapter 7, “Product Attributes with Business Component Domains.”

Domain Data Types

The data type you specify in the attribute definition determines how the application interprets the values in the domain. For example, you define an attribute with a list of values domain. You define the attribute values to be 1, 5, 10. To write configuration rules that perform numeric computations using these values, you must select the data type Integer or Number when defining the attribute.

The domain of an attribute can be one of the following data types:

- **Boolean.** Use this data type when the user’s input is true or false, yes or no. If you specify the Integer data type for these inputs, the application assigns 1 for True or Yes inputs. False and No are assigned 0.

- **Number.** The attribute value can be any positive or negative real number. In Boolean expressions, numbers greater than 0 are interpreted as true. Omit commas when specifying the domain. For example, enter 10,000 as 10000.

  NOTE: When you specify a number, make sure that the number of fractional digits is consistent with the regional or locale setting, defined in the Administration - Data, then the Locale view. If you do not, rules written against the attribute may not behave as anticipated, because the UI displays the number based on the regional or locale setting.

- **Integer.** The attribute value can be any positive or negative whole number. If a computation results in a fractional amount, the result is rounded to the nearest whole number. In Boolean expressions, integers greater than 0 are interpreted as true. Omit commas when specifying the domain. For example, enter 10,000 as 10000.

- **String.** The attribute value can be letters, numbers, or any combination. Attributes with this data type cannot be used as operands in a computation or as the result of a computation. The only arithmetic operator that can be used with this data type is = (equals). For example, you can write rules that test if the user has picked a specific string from a list of values.

- **Date.** The attribute value is interpreted as a date and must be in the correct date format. The system administrator sets date format defaults. Arithmetic computations using dates is not supported. For example, you cannot increase or decrease a date using a computation. All comparison operations are supported for dates. For example, you can compare two dates and determine whether one is earlier than (<), later than (>), or the same as (=) another date. Data type mismatches cause the user's input to be rejected, or can cause indeterminate results. For example, comparing a date data type to an integer data type.

About Hidden Attributes

When you place a check mark in the Hidden field in an attribute definition, the attribute does not display in the Quote, Order, Agreement, or Asset views or customizable product selection pages. For example, if you assign a product to a product class that has hidden attribute A1. When you add this product to a quote and select Dynamic Attributes, A1 does not display.

Use hidden attributes to create configuration parameters that customers do not need to see. For example, you could define a hidden attribute whose value is the number of bays required for a chassis. You could then write configuration rules that use the value of this attribute to monitor the number of available bays during a configuration session.

Upgrade users. Use hidden attributes as a replacement for virtual products.

Scenario for Creating Products with Attributes

This topic gives one example of how attributes may be used. You may use attributes differently, depending on your business model.

A business sells work shirts that have the following attributes:

- All brands of shirts come in the sizes S, M, L, XL.

- All brands of shirts come in the colors tan, green, blue, and brown.
- Some brands of shirts give the customer the option of personalizing the shirt by adding the company name.

Customers must pay extra for the XL size work shirt and for personalized work shirts.

To set up these attributes, the product administrator:

- Creates attribute definitions:
  - The attribute Work Shirt Size has the domain S, M, L, XL.
  - The attribute Work Shirt Color has the domain tan, green, blue, brown.
  - The attribute Work Shirt Personalization has the domain Y, N.
- Creates product classes:
  - The product class Work Shirt is associated with the attributes Work Shirt Size and Work Shirt Color.
  - The product class Personalized Work Shirt is a subclass of the product class Work Shirt. From the product class Work Shirt, it inherits the attributes Work Shirt Size and Work Shirt Color. It is also associated with the attribute Work Shirt Personalization.
- Associates all its work shirt products with these product classes:
  - If the work shirt cannot be personalized, it is associated with the product class Work Shirt.
  - If the work shirt can be personalized, it is associated with the product class Personalized Work Shirt.
- Sets up attribute pricing to reflect the extra cost of the XL and personalized work shirts.

Process of Creating Products with Attributes

To create products with attributes, perform the following tasks:

- "Creating Attribute Definitions" on page 66. First you create attribute definitions, with the domain for each attribute.
- "Creating Product Classes in a Hierarchy" on page 67. Then you create the hierarchy of product classes. For each product class, you select attribute definitions that you created in the previous step.
- "Associating Attributes with a Product" on page 69. Then you associate products with product classes. The products inherit all the attributes of the product class.
- "Setting Up Attribute Pricing" on page 70. If these attributes affect price, you must set up attribute pricing.

Creating Attribute Definitions

First, you create definitions of all the attributes that will be associated with your product.

NOTE: If you change the definition of an attribute by modifying its domain, the change applies only to versions of the product released after you changed the attribute.

This task is a step in "Process of Creating Products with Attributes" on page 65.

To create an attribute definitions

1. Navigate to the Administration - Product screen, then the Attribute Definitions view.
2. In the Attribute Definitions list, add a new record.
3. In the Attribute field of the new record, enter a name for this attribute.

   The Locked Flag is automatically selected.
4. In the Versions list, click the Work Space version.
5. In the Details list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Data Type	Select the data type for the domain of this attribute. For more information, see "About Attribute Domains" on page 62.
Domain Type	Select the type of the domain for this attribute. Options are Free Form and Enumerated. For more information, see "About Attribute Domains" on page 62.
Unit of Measure	Optionally, select the unit of measure for this attribute, such as day, month, dollar, dozen.

6 If you selected Enumerated as the domain type, you must enter all the values for the domain by adding new records to the Attribute Values list and completing the necessary fields, described in the following table.

Field	Comments
Value	Enter a valid value for the attribute.
Sequence	Enter a value to control the order of evaluation of constraints for attributes and relationships.

NOTE: If you enter a number in the Value field containing digits to the right of a decimal, these digits might automatically round up in the field display. For example, if you enter the number 12.349, the field might display 12.35. If digits automatically round up in the field display, an error results when you customize a product that uses this attribute, and you cannot select this attribute. Number rounding occurs if the Data Type field is Number, the Domain Type field is Enumerated, and the Number Fractional Digits field for your locale (in the Administration - Data screen, then the Locale view) is set to a number less than the number of digits that you enter to the right of the decimal. To avoid number rounding and the consequent error, increase the number in the Number Fractional Digits field and restart the server.

7 When you are finished, deselect the Locked Flag.

Creating Product Classes in a Hierarchy

Next you create product classes. This task is a step in "Process of Creating Products with Attributes" on page 65.

You create and manage product class hierarchies by specifying a parent product classes when defining product classes. Subclasses inherit the domains of their parent product classes. For more information, see "About the Product Class Hierarchy" on page 61.

You can view the hierarchy in by clicking the Structure view tab. This view contains a tree display that shows the hierarchy in a manner very similar to the Microsoft Windows file Explorer. You can expand or collapse classes and subclasses as needed to view the hierarchy. The portion of the hierarchy in which you are located displays in the Classes list.

To create a product class

1 Navigate to the Administration - Product screen, then the Product Classes view.

2 In the Product Classes list, add a new record, and enter a name for the class in the Product Class field.

The Locked Flag is automatically selected

3 In the Versions record, complete the necessary fields. Some fields are described in the following table.

Field	Comments
Parent Product Class	If this product class is a subclass, select the parent product class. This controls the product class's location in the class hierarchy.
Searchable	Select this field to make the product class available for parametric search.

4 In the Versions list, click the name of the Work Space version.

5 Click the Display Name view tab.

6 Add a record to the Display Name list, for each language that the product will be displayed in, and complete the necessary fields, described in the following table.

Field	Comments
Display Name	Enter the name that will be seen by customers using the application in this language.
Language	Select a language code, such as ENU for American English.

7 Click the Attributes view tab.

8 In the Attributes list, add new records for all the attributes in this product class, and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter a name for this attribute
Attribute Definition	Select the definition for this attributes. Definitions are available if they were created as described in "Creating Attribute Definitions" on page 66.
Default Value (Display)	Enter the default value that the end user sees when the product is initially displayed. The attribute has this value unless the end user selects a different value. However, if you write rules that manipulate the attribute value, the Siebel Configurator engine can override the default value. **NOTE:** Fields that include (Display) in their names are translations of the value in the administrator's location. Fields without (Display) in their names are in the default or Language Independent Code (LIC) location.

Field	Comments
Rejected Value (Display)	This field is relevant when an attribute is marked as required. If the attribute is required, the value in this field is the value that Siebel Configurator interprets to be not valid. The attribute will be marked as not entered, if this is the attribute's value. To use this checkbox, follow the instructions in "Setting Up Required Attributes" on page 70.
Required	Select this checkbox to require the user to select a value for this attribute. To use this checkbox, follow the instructions in "Setting Up Required Attributes" on page 70.
Read Only	Select this check box to make this attribute read-only, so the user cannot change the default value.
Hidden	Select this check box to prevents the attribute from displaying in quote, agreement, order, or asset views or customizable product selection pages.
Unit of Measure	Optionally, select the unit of measure for this attribute, such as day, month, dollar, dozen.
Searchable	Select this checkbox to allow this attribute and its values to be used in parametric searches. For example, if the attribute is Color, you can search for products that have Color = Red.
Inherited	This checkbox is selected if the attribute value is inherited from a parent product classes. Read-only.
Modified	This checkbox is selected if properties of the inherited attribute such as Read-only, Default, and Rejected were overridden by the user.

9 Click the Structure view tab and verify that this product class displays in the correct location in the class hierarchy.

10 When you are finished, release the Product Class record, so other users can work on this product class.

The new product class definition appears in the Product Classes list.

Associating Attributes with a Product

To associate attributes with a product, you assign the product to a product class. The product inherits all the attributes of the product class or subclass to which it is assigned.

This task is a step in "Process of Creating Products with Attributes" on page 65.

NOTE: Before you associate the product with a product class, you must create the product, as described in Chapter 3, "Basic Product Administration."

To associate a product with a product class

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Versions list, in the record for the Work Space version, in the Product Class field, select the product class.

Setting Up Attribute Pricing

If these attributes affect price, you must set up attribute pricing. For example, if you charge more for products in the largest sizes or for products of certain colors, you must set up pricing for the attributes that customers pay extra for.

For more information, see the topic about attribute pricing in *Siebel Pricing Administration Guide*.

Setting Up Required Attributes

You can set up attributes to require users to select values for them.

If you do not enter values in the mandatory fields where attributes are required, then Siebel Configurator displays an error message.

If you click Finish It!, Siebel Configurator does not complete required attributes for the user, as it completes required cardinalities. Instead, it displays an error message saying that the configuration is incomplete.

To set up required attributes, follow these steps:

- Set up the attribute values as usual. If it is an enumerated attribute, include in the list of possible values a value that would not be acceptable for this attribute. This value must match the type of the attribute, for example 999 in the type is numeric. Give this value a display name that will be displayed as an error message and that will be displayed in the control telling the user to select a value; for example, give it the display name *Select a Color*. For more information about setting up attribute values, see “Creating Attribute Definitions” on page 66.
- Attach the attribute to a product class, and select the Required field. As the rejected and default value for the product class, enter 999, the value for the message. Because it is the default value, it is displayed in the drop-down control for the attribute. For more information about attaching an attribute to a product class, see “Creating Product Classes in a Hierarchy” on page 67.

  NOTE: As an alternative to designating an attribute as required in the product class definition, you can designate it as required in the product definition after the attribute has been attached to the product through the product class assignment. This gives the administrator the flexibility to have the same attributes on a product class required on some products and not required on other products.
- Attach the product class to the product. The product inherits the rejected value from the product class. For more information about attaching a product to a product class, see “Associating Attributes with a Product” on page 69.

Setting a Read-Only Value for an Attribute of a Product

When you set the value of an attribute for a product, it cannot be changed by either the user, a configuration rule, or the Siebel Configurator engine. One example, is when you want to set an attribute value so that provide and consume rules can use it to add or subtract from a defined resource.

For example, you create an attribute called Slots Required for a product class containing expansion cards. Some cards take up one expansion slot; some take up two. You could define a list of values containing the integers 1 and 2 and make it the domain for Slots Required. For each expansion card you would then set the value of this attribute at 1 or 2. Users cannot change this value when configuring the product, and configuration rules cannot change this value.

You would then write a provide rule that increases a Slots Available resource when the user picks a chassis. For the expansion card product class, you would write a consume rule that reduces Slots Available by the value of Slots Required, each time the user picks an expansion card. In this fashion, you use attribute values as constants that interact with a defined resource to manage a consumable configuration variable.

To set an attribute value for a product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Attributes view tab.

   The Attributes list appears, displaying all the product's attributes inherited from its product class or subclass.
5. In the record for the desired attribute:
   a. Select the desired attribute value in the Default Value field.
   b. Select the Read Only checkbox.

Creating a Siebel Configurator Engine Read-Only Attribute

When you create an attribute that is Siebel Configurator engine read-only, it cannot be changed by the Siebel Configurator engine or a configuration rule. Siebel Configurator engine read-only attributes can only be changed by the user. This feature enables users to define attributes that can serve as an input to compute a valid solution but will not be updated as part of computing the solution. You can create a Siebel Configurator engine read-only attribute using the ConstraintReadOnly UI property. One example, is when you do not want the Siebel Configurator engine to overwrite specific attribute values.

For example, you create an attribute called Age to capture the user's age that defaults to an empty string. You would then write a rule that excludes the Under 18 Insurance Charge product when the Age attribute is greater than 18.

If the user sets the Age attribute value at 25, then the product Under 18 Insurance Charge appears in red in the UI because it is excluded. When the user tries to select the Under 18 Insurance Charge product, the application displays a configuration conflict error message. If the Age attribute is set to ConstraintReadOnly = Y, the UI does not display a Proceed option and the age attribute remains at 25 and the user cannot select the Under 18 Insurance Charge product.

If the Age attribute is set to ConstraintReadOnly = N, the UI displays the Proceed option. The proceed option selects the Under 18 Insurance Charge product and resets the Age attribute to a null value. The Age attribute value that is retrieved by Siebel Configurator will serve as a static input to compute a valid solution. If a valid solution cannot be found, then the engine will generate an error message without a Proceed option. The absence of the Proceed option prevents Siebel Configurator from overwriting the value of the Age attribute.

To create a Siebel Configurator engine read-only attribute:

1. Navigate to the Administration - Product screen, and then Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Properties view tab.
5. In the explorer display of the product on one side of the screen, select the desired read-only attribute.
6. In the Properties list, add a new record and complete the fields. Some fields are described in the following table.

Field	Comments
Name	Enter ConstraintReadOnly.
Value	Enter a value for the property. The value is either Y or N.

Creating a Configuration Session Specific Computed Attribute

You can define an attribute that is specific to the configuration session. The attribute value is computed during the start of the configuration session, but it is not synchronized to the line items or the database. The attribute is treated as a virtual attribute for a product and is accessible within the Siebel Configurator user interface, constraints, scripts and properties.

One example is when you want to capture an attribute value whose sole purpose is to trigger a product's constraints. For example, you create an attribute called Additional Fee with a domain value of Yes and No, where the default value is set to No. You then write a requires rule that selects products Late Fee and Finance Charge when Additional Fee = Yes. When the user launches Siebel Configurator, and sets the Additional Fee = Yes, the constraints are triggered and Additional Fee products are selected. Note that when you exit from Siebel Configurator, you will not be able to see the Additional Fee attribute displayed in the line item attribute applets. Thus reducing the number of attributes that are captured in the database primarily for computational purposes.

NOTE: It is recommended that you do not reference these attributes within Eligibility & Compatibility and pricing rules. You also cannot set the Required field to Y within the attribute definition of the class or product.

To create a configuration session specific computed attribute

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Properties view tab.
5. In the explorer display of the product on one side of the screen, select the name of the root product.
6. In the Properties list, add a new record and complete the fields. Some fields are described in the following table.

Field	Comments
Name	Enter NoSave.
Value	Enter a value for the property. The value is either Y or N.

7. In the explorer display of the product on one side of the screen, select the attribute.
8. Enter the same information for the attribute that you entered for the product in Step 6.

Binding an Attribute Value to a Line Item Integration Component Field

You can create an attribute and bind the value to a quote, an order, or an asset item field. When you start the configuration session, the value from the business component field is loaded into the product instance's attribute value. When you exit the configuration session, the attribute value is copied to the target business component field value. This attribute is accessible in the Siebel Configurator user interface, constraints, scripts, and properties. If there is any data inconsistency between the field data and the product instance, then a warning window displays at the start and end of the configuration session. Note that the business component field that is mapped to the attribute value must exist in the corresponding line item integration components for quotes, orders or asset. This streamlines the order capture process by enabling users to provide line item characteristics within Siebel Configurator thereby preventing users from writing constraints or scripts to enable back and forth data exchange between attribute values and line item fields.

For example, you may want to create an attribute value to represent the Billing Account. The Billing Account is tracked in the line item field. The product administrator sets the property LineItemICField = Billing Account for the Billing Account Attribute.When the configuration session is started, the Billing Account attribute is defaulted to the business component Billing Account field. The end user has the option to overwrite the defaulted value. When the user exits Siebel Configurator, the user's selection is copied back to the business component.

NOTE: When you bind an integration component field of type Currency, you must use the Maximum List column of the Attribute Definition applet. This prevents the user from exceeding the maximum value of the Currency field for the business component field located in the integration component field.

To bind an attribute value to a line item

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Properties view tab.
5. In the explorer display of the product on one side of the screen, select the desired attribute.
6. In the Properties list, add a new record and complete the fields. Some fields are described in the following table.

Field	Comments
Name	Enter LineItemICField.
Value	Enter a value for the property. The value is the LineItem field name. **NOTE:** If the LineItemICField is mapped to a Currency field, then this field has a limit of 10 digits.

Defining a UI Property Using an Attribute Value from a Line Item Integration Component Field

After you bind an attribute value to a line item, you can use that attribute value in the Siebel Configurator user interface, constraints, scripts, and properties.

This section shows you how to use the value to define a user interface property. There is general information about defining a user interface property in "Defining a UI Property" on page 182. If the value is from an integration component field, you must also do the following:

- The Line Item fields must be present in the Line item Integration Components and in the SIS OM PMT Service.
- If asset-based ordering is enabled, you must modify the following integration objects: SIS OM Quote, SIS OM Order, SIS Order Asset
- If asset-based ordering is not enabled, you must modify the following integration objects: 7.7 Asset Integration Object, 7.7 Quote Integration Object, 7.7 Order Entry Integration Object

Then you can define the UI property using the following procedure.

To define a UI property using an attribute value from a line item integration component field

1. In Web Tools, open a workspace and then navigate to Object Explorer.

   For more information on using the workspace dashboard, see *Using Siebel Tools*.
2. Add custom fields to the Line Item Integration components and SIS OM Payment Service.
3. Deliver the workspace.
4. In the Siebel Business Application, bind the line item field to the attribute.

   For more information, see "Binding an Attribute Value to a Line Item Integration Component Field".

Changing Inherited Properties of Attributes

When you associate an attribute with a product class, it is inherited by all member subclasses. If you edit an attribute on the product class where it was originally defined, the changes propagate to all member subclasses. The attribute definition is uniform for all subclasses that inherit it.

Subclasses can have two kinds of attributes: local and inherited. A local attribute is one that is defined on the subclass. An inherited attribute is one that is inherited from a parent product class.

You customize an inherited attribute domain by editing its definition at the subclass level. When you edit an inherited attribute definition, the changes propagate to all members of the subclass, including other subclasses under that subclass.

Editing an inherited attribute permanently breaks attribute inheritance for the fields you edit. Editing the domain of an inherited attribute permanently prevents an attribute from inheriting domain changes from its parent attribute.

If you delete the parent product class attribute, it is not deleted from subclasses where inheritance is broken. (The attribute definition is deleted from all subclasses where inheritance has not been broken.)

For example, you have the class hierarchy in Figure 2. Product Class A has one subclass called Subclass B. Subclass B has one subclass called Subclass C. Class A has Attribute A defined on it. Subclass B has attribute B defined on it. Subclass C has Attribute C defined on it. Subclass B inherits Attribute A from Class A. Subclass C inherits Attribute A from Class A and Attribute B from Subclass B.

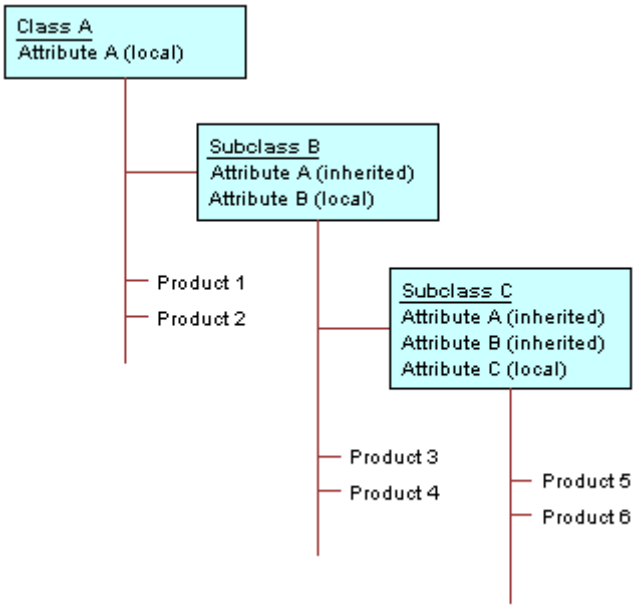

Figure 2. Attribute Inheritance

In Subclass B, you edit the domain of Attribute A by entering a new list of values and default Value. Subclass B no longer inherits changes to these fields from Attribute A in Class A, its parent attribute.

When you edit a local or inherited attribute, the changes propagate to all members of the product class or subclass. In the example, the new values propagate to Attribute A in Subclass C.

There are restrictions on which fields you can edit for inherited attribute properties. These restrictions are shown in Table 3.

Table 3. Editable Fields in a Subclass Inherited Attribute Properties

Field	Editable?
Name	Yes. Breaks inheritance for all fields. Same as defining new attribute.
Data Type	Yes. Breaks inheritance for all fields. Same as defining new attribute.

Table 3. Editable Fields in a Subclass Inherited Attribute Properties

Field	Editable?
Default Value	Yes. Breaks inheritance for this field.
Required	Yes. Breaks inheritance for this field.
Display Name	Yes. Breaks inheritance for this field.
Parametric Search	Yes. Breaks inheritance for this field.
Unit of Measure	Yes. Breaks inheritance for this field.
Description	Yes. Breaks inheritance for this field.

To change an inherited property of an attribute

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Attributes view tab.
5. In the list applet, select the attribute you want to change.
6. In the desired record in the Attributes list, change the property of the attribute.

   You can change all the fields except Name and Attribute Definition.

Viewing Changes in the Inherited Properties of Attributes

You can view changes that users have entered to override the inherited properties of an attribute.

To view changes in inherited properties of an attribute

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Attributes view tab.
5. In the list applet, select the attribute whose changes you want to view.
6. From the Attributes list menu, select Show User Input.

   A text file appears, with a list of changes that users made to the inherited properties.

Restoring the Inherited Properties of an Attribute

If users have changed the inherited properties of an attribute, you can discard all these changes and restore the inherited properties.

To restore the inherited properties of an attribute

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Attributes view tab.
5. In the list applet, select the attribute whose properties you want to restore.
6. From the Attributes list menu, select Restore.

Changing the Hidden or Required Settings for a Product Attribute

When you define an attribute at the class level, you can set the attribute to be hidden or required:

- Hidden attributes do not display in the Quote, Order, Agreement, or Asset views.
- Required attributes are those where the user must select a value for the attribute. The value of the attribute cannot be blank.

Attribute definitions propagate automatically to all the products that belong to the product class. However, you can change the Hidden flag and the Required flag settings for an attribute at the product level. This lets you manage the hidden or required settings for attributes product by product.

You can use the hidden setting to simplify your product class system. For example, if a product class has 8 attributes and a product has 7 of these attributes, you can put the product in this class and hide the eighth attribute. You do not have to create a special subclass with 7 attributes for the product.

To change the hidden or required settings for attributes of a product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Click the Attributes view tab.
5. In the Explorer applet, select the attribute you want to hide.
6. In the Attributes list, in the record for the attribute, select the Hidden checkbox.

To change the hidden or required settings for attributes of a product class

1. Navigate to the Administration - Product screen, then the Product Classes view.
2. In the Product Classes list, select and lock the desired product class.
3. In the Versions list, click the Work Space version.
4. Click the Attributes view tab.

5 In the Attributes list, in the record for the attribute, select the Hidden checkbox.

About Managing Product Classes

You can manage product classes in the following ways:

- "Viewing Related Objects for Product Classes" on page 79
- "Editing a Product Class Definition" on page 79
- "Deleting Product Class Records" on page 81
- "Exporting or Importing Product Classes" on page 81

Viewing Related Objects for Product Classes

The Product Class Cross References view allows you to view the objects that are related to a product class, including:

- **Subclasses.** Classes that are a child to the product class
- **Products.** Products that have the product class as a parent product class
- **Relationships.** Objects that use the product class to form relationships
- **Attribute Adjustments.** Attribute Adjustments that use the product class

The user can drill down on any of these line item to display the administration view for that object.

NOTE: The Cross References list applies to one version of a product class, because different versions may include different direct subclasses. Versions include Workspace versions.

To view related objects for a product class

1 Navigate to the Administration - Product screen, then the Product Classes view.

2 In the Product Classes list, select the class whose related objects you want to view.

3 In the Versions list, click the name of version whose related objects you want to view.

4 Click the Cross Reference view tab.

5 In the Cross Reference link bar, click Subclasses, Products, Relationship, or Attribute Adjustments to view lists of related objects.

Editing a Product Class Definition

Editing a class definition record does not change the attributes defined on the class. However, if you change the parent class name of a subclass to another already-existing class name, this changes the location of the subclass in the class hierarchy and can change which attributes the products in the subclass inherit.

For example, a subclass SC1 has parent class PC1, which has three attributes defined on it A1, A2, A3. This means SC1 inherits attributes A1, A2, A3. Class PC2 has attributes A4, A5, A6 defined on it. If you change the parent class of subclass SC1 from PC1 to PC2, this changes the attributes inherited by SC1 to A4, A5, A6. You have moved SC1 from being a subclass of PC1 to being a subclass of PC2.

If you are changing the parent-class name for class, do the following first.

To prepare a product class for a parent-class name change

1. Run a query in the Products list to identify all the products assigned to the class.
2. Analyze how changing the parent class name of the class will affect the attributes inherited by these products.
3. Identify all pricing rules defined for the attributes inherited by the class.

   Note which rules must be changed to reflect the new parent class name and any new attributes.
4. Identify all configuration rules that refer to inherited attributes of the class.

   Note which rules must be changed to reflect the new parent class name and any new attributes.

If you are changing the class name, do the following first.

To prepare a product class for name change

1. Identify all pricing rules defined for the attributes inherited by the class.

   Note which rules must be changed to reflect the new parent class name and any new attributes.
2. Identify all configuration rules that refer to inherited attributes of the class.

   Note which rules must be changed to reflect the new parent class name and any new attributes.
3. Identify all product with components relationships of type Class and Dynamic Class that have been defined using the class.

   Note which relationships need to be redefined to reflect the new class name.
4. Identify any customizable product UI properties defined for the class.

   Note any UI property definitions that must be revised to reflect the new class name.

If you are changing a class definition, do the following first.

Before editing a class definition, make sure you have fully analyzed the impact on attribute inheritance.

Also make sure you have analyzed the impact on pricing rules, configuration rules, and UI design.

To edit a class definition

1. If you are changing the parent class name of a class, verify that all the steps in preparing the class for a parent-class name change are complete. You must do this using the Work Space version of the parent class.

2 If you are changing a class name, verify that all the steps in preparing a class for name change are complete.

3 Navigate to the Administration - Product screen, then the Product Classes view.

4 In the Product Classes list, select and lock the desired record.

5 In the Versions list, click the Work Space version.

6 Modify fields, user interface definitions, constraints, and other information as needed.

NOTE: These changes will not take effect until you release the new version by selecting this class and clicking Release in the Administration - Products screen, then the Product Classes view.

Deleting Product Class Records

You cannot delete a product class record. If you no longer want to use the product class, you can deactivate all versions rather than deleting the product class.

Before doing this, verify that the product class is not used for defining any active products.

If you have a large number of inactive versions for a given product class, you can delete the versions using the CleanupSingleObject method of the ISS Authoring Import Export Service. For more information, see *Siebel Order Management Guide*.

To expire or deactivate a product class

1 Navigate to the Administration - Product screen, and then the Product Classes view.

2 In the Product Classes list, select and lock the desired product class.

3 In the Versions view, deselect the Active checkbox for the Work Space version.

4 Release the Work Space version.

Exporting or Importing Product Classes

You can export a product class or the whole product class structure to another database. When you export a product class, the following parts are included in the export:

- The parent product class of the product class you are exporting plus all the subclasses of the parent product class. When you export a product class, the export contains not just the product class you selected, but the portion of the product class structure to which it belongs.
- Attribute definitions for the product classes and all subclasses.
- List of values definitions associated with attribute definitions. List of values are exported in the current language only.

The products in the product classes are not exported.

When you export the whole product class structure, all product classes and subclasses are exported, along with the items previously listed. Products are not exported.

When you export a product class or the product class structure, an XML file is created in a location you specify. The XML file contains the exported product class structure. When you import this product class structure, the application reads the XML file and synchronizes the product class system of the import database to the XML file. The XML file takes precedence, and the product class system is modified to reflect the portion of the product class system in the XML file.

For example, in the XML file the subclass shoes, has the parent product class footwear. In the import database the subclass shoes has the parent product class Wardrobe. After importing the XML file, the subclass shoes will have the parent product class footwear.

Use the following process to update the product class structure in database B with changes from database A.

- Back up database B.
- Export the desired product classes from database A.
- Import the product classes to database B.
- Compare the updated product class structure and list of values definitions in database B with database A.
- Verify that components in affected products with components in database B have the correct attributes.

Use the following process to update both the products and product class structure in database B with changes from database A:

- Use the previous process to update the product class structure in database B.
- Export the products from database A, except products with components.
- Import the products into database B. Verify that the products are in the correct product classes and inherit the correct attributes.
- Export products with components from database A.
- Import products with components to database B. For each product with components, verify that the component products are present and have the correct attributes.

To export a product class or the whole product class structure

1. Review the processes listed in the previous section.
2. Navigate to the Administration Product screen, then the Product Classes view.
3. In the Product Classes list, select the product class you want to export.
4. In the Versions list, select the version you want to export.
5. From the Versions menu, choose Export Version.

   The Export Versioned Object dialog box appears.
6. In the dialog box, click Object(s) Only to export the product class or click Full Structure to export the whole product class structure.

   A Save As dialog box appears.

7 Browse to the location where you want to store the file, specify the file name, and then click Save.

The application creates an XML file containing the exported product class structure and stores it at the location you specified.

When you import a product class structure, you must import the entire contents of the export file. You cannot choose which product classes in the file to import.

To import a product class structure

1 Review the processes above.

2 Navigate to the Administration - Product, then the Workspace Projects view.

3 Select the desired workspace record.

4 From the applet-level menu, select Import Contents.

The new product class structure is imported into the database.

About Managing Attribute Definition Records

You can manage attribute definition records in the following ways:

- "Viewing Related Objects for Attribution Definitions" on page 83
- "Editing Attribute Definitions" on page 84
- "Deleting Attribute Definitions" on page 84

Viewing Related Objects for Attribution Definitions

The Attribute Definition Cross References view allows you to view the objects that are related to an attribute definition, including:

- **Class.** Product classes that contain the Attribute Definition
- **Smart Part Number.** Smart Part Numbering that makes use of the Attribute Definition

To view related objects for an attribute definition

1 Navigate to the Administration - Product screen, then the Attribute Definitions view.

2 In the Attribute Definitions list, select the definition whose related objects you want to view.

3 In the Versions list, click the version whose related objects you want to view.

4 Click the Cross References view tab.

5 In the Cross References link bar, click the Class or Smart Part Number link to view lists of related objects.

Editing Attribute Definitions

When you edit attribute definitions, you modify the attribute domain.

To edit an attribute definition

1. Navigate to the Administration - Product screen, then the Attribute Definitions view.
2. In the Attribute Definitions list, select and lock the desired attribute definition.
3. In the Versions list, click the Work Space version.
4. In the Details view, edit the definitions as necessary.

Deleting Attribute Definitions

You cannot delete an attribute definition. If you no longer want to use the attribute, you can deactivate all versions

Before doing this, verify that the attribute is not used in any configuration rules, or for attribute-based pricing.

To expire or deactivate an attribute definition

1. Navigate to the Administration - Product screen, then the Attribute Definitions view.
2. In the Attribute Definitions list, select the desired attribute definition.
3. In the Versions view, deselect the Active checkbox for the Work Space version of the attribute definition.
4. Release the Work Space version.

Viewing Product Attributes

If a product has been assigned to a product class, it inherits all the attributes defined on the product class. You can verify that you have defined product classes and associated them with products correctly by viewing the attributes for the products.

To view a product's attributes

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the desired product.
3. In the Versions list, click the name of the desired version.
4. Click the Attributes view tab.

7 Product Attributes with Business Component Domains

This chapter describes how to define attributes that have a business component domain. These attributes allow users to select a record from a pick applet, also called a dialog box. A field in this record then displays in the selection page as the value of the attribute.

This chapter requires that you be familiar with creating pick applets in Oracle's Web Tools.

This chapter includes the following topics:

About Attributes with Business Component Domains

The Product Administrator has created a product with components called Premier Service Package. The Product Administrator wants users to be able to select an account name when configuring the product.

This product has been assigned to a product class that has the attribute Account defined on it. Account has been added to a group in the User Interface view and will display in a selection page.

In the selection page, the Account attribute displays with a blank text field and a select icon. When the user clicks on the select icon, a dialog box displays containing available accounts. When the user selects an account, the account name is transferred to the Account text box.

In this scenario, the user was able to access a Siebel business component to display account records. When the user selected an account record, the account name field in the record was transferred to the selection page and became the value of the Account attribute. The domain of the Account attribute is the records retrieved by the business component and displayed in the dialog box, also called a pick applet.

Attributes with a business component domain differ from attributes with other domain types in several ways:

- The attribute values are not defined by a list of values. The user selects the attribute's value directly from a pick applet, which displays information from a business component.
- Attributes can be defined so that when the user selects the value for one attribute (the primary attribute), the values for other attributes are automatically selected. For example, if the user selects a value for Account Name (the primary attribute), the value of the Address attribute is filled in automatically. Configuration rules can be written only on the primary attribute. You cannot write configuration rules on attributes whose values are automatically selected based on the value of the primary attribute.

About the UI Properties for Attributes with Business Component Domains

You can define an attribute that has values the user can select from a pick applet. The pick applet displays records from a specified business component. When the user selects a record, a specified field in the record displays in the selection page as the value of the attribute.

Several predefined UI properties are provided to associate the attribute with a pick applet and picklist. Defining these UI properties on the attribute replaces Web Tools procedures for configuring the originating business component when defining a pick applet.

You define these UI properties on the attribute that you have set up to display a select button in the selection pages. These UI properties associate the attribute with a pick applet and a picklist and are shown in Table 4.

Table 4. Predefined UI Properties

Name	Value
PickApplet	The name of the pick applet.
PickList	The name of the picklist.
PickMap01	This is an XML tag that associates the attribute name with the picklist field that you want to display.
PickMap*nn*	PickMap02 and so on, display multiple fields from the same picklist. You can also use this PickMap definition to define a constraint on the records the user sees in the pick applet. Note that constraints specified in the pickmap definition cannot include search expressions. If you need to define a constraint with an expression that will be computed at runtime and is dynamic in nature, then use the PickSpec UI property.
PickSpec01	Specify the search specification expression that constrains the records that the user sees on the pick applet. The current instance path can be used as a value in the search specification. The PickSpec value is a search expression that uses the syntax for the BC Search Specification defined in Oracle Web Tools and the standard Siebel Configurator path syntax to define instance variables and values. For example, you can define a constraint to show contacts over the age of 18 when the currently selected product's account type attribute is set to primary. See Table 8 for details.
PickSpec*nn*	PickSpec02 and so on, specify multiple search specifications on the same pick applet. Note that the search specification must be valid. To verify this, use the BC search specification in Web Tools.

The PickMap value is an XML tag that has the following format:

<PickMap Field="AttributeName" PickField="BusCompFieldname" Constrain="Y/N" BusObj="BusObjName" />

- **Field.** For UI properties with name PickMap01, this specifies the name of the attribute in the selection page. Use the attribute name, not its display name.

  For UI properties with Constrain="Y", it specifies the business component field name or attribute name to be used as a filter for the records displayed in the pick applet. The format for specifying a business component is `buscompname.fieldname`. The format for specifying an attribute name is `attributename`.

  For example, to constrain the records displayed in the pick applet to those having the current record's Opportunity name in the Quotes view, (Quote business component), you would enter `Quote.Opportunity`. To constrain records to the value of the Account Name attribute, you would enter `Account Name`.

- **PickField.** Specifies the picklist field name. When the user selects a record from the pick applet, the contents of this field becomes the attribute value shown in the selection page. Use the business component field name, not the field's display name.

- **Constrain.** When set to "Y" (Yes), PickMap specifies a business component field that filters the records the user sees in the pick applet. If not specified, the default is "N" (No).

- **BusObj.** Specifies the business object in which the PickMap definition is active. If omitted, the PickMap definition is active in all business objects. For example, if you set BusObj="Order Entry", the PickMap applies to orders but not quotes. Use this argument to constrain the pick applet differently for orders than for quotes. If you insert a BusObj argument in PickMap01, this limits the display of the select icon and pick applet to the specified business object, for example Quote.

Process of Creating an Attribute with a Business Component Domain

You associate a business component with an attribute using the same process as creating a pick applet in Oracle's Web Tools, with the following modifications:

- **Configuring the originating applet.** The selection page takes the place of the originating applet. You replace this procedure with steps that define the attribute and insert it in the selection page. To do this step, see "Adding the Attribute to a Selection Page" on page 88.

- **Configuring the pick applet.** You can use an existing pick applet or define a new applet. If you define a new pick applet, there is no change to the procedures described in *Siebel Tools Reference*.

- **Configuring the originating business component.** These procedures are replaced by defining a series of UI properties on the attribute. These UI properties specify the pick applet name, picklist name, and pick map definitions. You can define multiple pick maps that display the content of several fields from the same record. You can also define UI properties to constrain the pick list.

- **Configuring the picklist.** You can use an existing picklist or define a new one. If you define a new one, there is no change to the procedures described in *Siebel Tools Reference*.

Assuming that the picklist and pick applet are already defined in Oracle's Web Tools, to define an attribute with a business component domain, perform the following tasks:

1. "Adding the Attribute to a Selection Page" on page 88. First, add the attribute to a selection page.
2. "Associating the Attribute with a Business Component" on page 89. Associate the attribute with a business component by defining UI properties on it.
3. "Setting Up Multiple Fields for Display" on page 90. Optionally, you can set up attributes so that selecting a value for one automatically selects the values for others.
4. "Creating a Business Component Field Constraint" on page 92. Optionally, you can constrain a pick applet so that it displays only the records having a specified field value.
5. "Creating an Attribute Value Constraint" on page 95. Optionally, you can constrain the records that display in the pick applet based on the value of an attribute in the selection pages.
6. "Creating a Search Expression Based on the Current Instance" on page 95. Optionally, you can constrain the records in a pick applet based on an expression that will be computed at run time and is dynamic in nature.
7. "Refreshing Attributes with Business Component Domains" on page 99. Optionally, you can refresh the attribute value with the business component field value when you launch Siebel Configurator.

Adding the Attribute to a Selection Page

This step creates the attribute in the class system and defines where it displays in the selection pages. It replaces configuring the originating applet step in the Oracle's Web Tools process for creating a pick applet.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

The attribute data type must be the same as the data type of the business component field from which the attribute value will come. For example, if the data type of the business component field is Boolean, the attribute data type must be Boolean. The application does not verify that the attribute data type and the business component field data type are the same.

In the User Interface view, you do not need to pick a UI control for the attribute. When you define the PickMap01 UI property on the attribute, the application automatically assigns a text field with select button to the attribute. When you select a record from the pick applet, the specified field in the record displays in the attribute text box.

If you select a UI control for the attribute, it will be overridden by the text box with select button.

To add the attribute to a selection page

1. Define an attribute on a class. Leave the list of values and the Default Value field blank.
2. Verify that only products that are themselves customizable products or will always be components of customizable products are assigned to the class. Products that will be part of bundles must not be assigned to the class since they are not configured using selection pages.

3 Navigate to the Administration - Product screen, then the Product Definitions view, and select and lock the desired product with components. This product must belong to the class on which the attribute is defined.

4 In the User Interface view, select a group and add the attribute to the Group Item List.

   Do not select a UI control for the attribute.

Associating the Attribute with a Business Component

This step defines the UI properties needed to associate the attribute to a picklist. It replaces configuring the originating business component step in the Oracle's Web Tools process for creating a pick applet. For information on these UI properties see "About the UI Properties for Attributes with Business Component Domains" on page 86.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

The Product Administrator has created a product with components called Premier Service Package. This product has been assigned to a product class that has the attributes Account, Location, and Opportunity defined on it. These attributes have been added to a group in the User Interface view and will display in selection pages.

The Product Administrator wants users to be able to select an account name when configuring the product. To do this, the Product Administrator must define the following three UI properties on the Account attribute:

- **PickList.** Its value is PickList Account.
- **PickApplet.** Its value is Account Pick Applet.
- **PickMap01.** This UI property provides the name of the attribute and the business component field. Its value is an XML tag that has the following elements:
  - PickMap Field = "Account". This is the attribute name.
  - PickField = "Name". This is the business component field.

The Account attribute displays with a text box in the configuration selection pages. When the user clicks the select button, the Account Pick Applet displays. When the user selects an account and clicks OK, the Account name is transferred to the Account field in the selection page.

Table 5 shows how to use the predefined UI properties to associate an attribute with a business component.

Table 5. UI Properties

Name	Value
PickApplet	The name of the pick applet.

Table 5. UI Properties

Name	Value
PickList	The name of the picklist.
PickMap01	This is an XML tag that associates the attribute name with the picklist field that you want to display. It can have the following values: ■ **Field.** Only the PickMap Field and PickField variables are required. Enclose their values in quotes. ■ **PickMap Field.** The attribute name in the selection page. ■ **PickField.** The picklist field to be used as the attribute value. The PickMap that provides this information must be named PickMap01.

Associating the attribute with a business component

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. Click the Properties view tab.
5. Select the desired attribute in the Explorer applet.
6. Click New in the List applet.
7. Define the UI properties as shown in Table 5.

Setting Up Multiple Fields for Display

You can group attributes together so that selecting a record from a pick applet for one attribute populates several attributes. You do this by defining additional PickMap UI properties on an attribute. These additional UI properties define how to populate the other attributes.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

The attribute on which you define PickMap01 is called the primary attribute. The user selects a value for this attribute and this causes the values for the other attributes to be selected automatically.

The Product Administrator has created a product with components called Premier Service Package. This product has been assigned to a product class that has the attributes Account, Location, and Opportunity defined on it. These attributes have been added to a group in the User Interface view and will display in selection pages.

The Product Administrator wants users to be able to select an account name when configuring the product. When they do, the Account Administrator wants to automatically populate the Location attribute with the state in which the account is located.

To do this, the Product Administrator must define the following UI properties on the Account attribute:

- **PickList.** Its value is PickList Account.
- **PickApplet.** Its value is Account Pick Applet.
- **PickMap01.** This UI property provides the name of the attribute and the business component field. Its value is an XML tag that has the following elements:
  - Field = "Account". This is the attribute name.
  - PickField = "Name". This is the business component field.
- **PickMap02.** This UI property defines an attribute that will receive its value automatically when the user selects a value for the primary attribute. In this case, the attribute is Location. The value of the UI property is an XML tag that has the following elements:
  - Field = "Location". This is the attribute name.
  - PickField = "State". This is the business component field.

The Account and Location attributes display with a text box next to them in the configuration selection pages. When the user clicks the select button and chooses an Account name, it is transferred to the Account field and the state name is transferred to the Location field.

Table 6 shows how to use the predefined UI properties to set up multiple fields for display.

Table 6. UI Properties

Name	Value
PickApplet	The name of the pick applet.
PickList	The name of the picklist.
PickMap01	This is an XML tag that associates the attribute name with the picklist field that you want to display. PickMap01 must be defined on the primary attribute Only the Field and PickField variables are required. Enclose their values in quotes. Field: The name of primary attribute. PickField: The name of the business component field.
PickMap02	Only the Field and PickField variables are required. These variables are for attributes other than the primary attribute. These attributes will be populated automatically when the user selects a pick applet record for the primary attribute. Attribute values for these attributes are read-only. Enclose the attribute values in quotes. Field: The name of the attribute, other than the primary attribute. PickField: The name of the business component field. You can define PickMaps to populate as many fields as desired. Number the PickMaps in sequential order, for example Pickmap03, PickMap04, and so on.

To set up multiple fields for display

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. Select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. Click the Properties view tab.
5. Select the desired attribute.
6. Define the UI properties as shown in Table 6.

Creating a Business Component Field Constraint

You can constrain the records that display in the pick applet based on a field that they have in common with the business component that starts the configuration session. This is similar to using Oracle's Web Tools to constrain the display of records in a pick applet based on a field in the originating business component.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

You do this by defining an additional PickMap UI property on an attribute. This additional UI property defines how to constrain the records in the pick applet specified in PickMap01. The constraint PickMap specifies the business component name and field to use to filter the records in the pick applet.

The Product Administrator has created a product with components called Premier Service Package. This product has been assigned to a product class that has the attributes Account, Location, and Opportunity defined on it. These attributes have been added to a group in the User Interface view and will display in selection pages.

The Product Administrator wants users to be able to select an account name when configuring the product in a quote. When they do, the Account Administrator wants to automatically populate the Location attribute with the state in which the account is located.

In addition, the Product Administrator wants to constrain the pick applet to display only the accounts associated with the opportunity name displayed in the Quote Opportunity field. For example, if the opportunity name is Boeing, the pick applet would display all the Boeing accounts only.

To do this, the Product Administrator must define the following UI properties on the Account attribute:

- **PickList.** Its value is PickList Account.
- **PickApplet.** Its value is Account Pick Applet.
- **PickMap01.** This UI property provides the name of the attribute and the business component field. Its value is an XML tag that has the following elements:
  - Field = "Account". This is the attribute name.
  - PickField = "Name". This is the business component field.

- **PickMap02.** This UI property defines an attribute that will receive its value automatically when the user selects a value for the primary attribute. In this case, the attribute is Location. The value of the UI property is an XML tag that has the following elements:
  - Field = "Location". This is the attribute name.
  - PickField = "State". This is the business component field.
- **PickMap 03.** This UI property filters the display of records in the pick applet to those having the same value as a field in the business component that starts the configuration session. The value of the UI property is an XML tag that has the following elements:
  - Constrain = "Y". This notifies the application that the UI property defines a constraint.
  - Field = "Quote.Opportunity". This is the business component name and field name that will be used as a filter.
  - PickField = "Name". This is the picklist field name that will be filtered.

The Account and Location attributes display with a text box next to them in the configuration selection pages. When the user clicks the select button for Account, a pick applet displays. It contains only the accounts that have the name specified in the Opportunity field of the quote that started the configuration session (Quote business component). When the user selects an account and clicks OK, the Account name is transferred to the Account field and the state name is transferred to the Location field.

Table 7 shows how to use the predefined UI properties to constrain the user's choices.

Table 7. UI Properties

Name	Value
PickApplet	The name of the pick applet.
PickList	The name of the picklist.
PickMap01	This is an XML tag that associates the attribute name with the picklist field that you want to display. PickMap01 must be defined on the primary attribute Only the Field and PickField variables are required. Enclose their values in quotes. Field: The name of primary attribute. PickField: The name of the business component field.

Table 7. UI Properties

Name	Value
PickMap02	Only the Field and PickField variables are required. These variables are for attributes other than the primary attribute. These attributes will be populated automatically when the user selects a pick applet record for the primary attribute. Attribute values for these attributes are read-only. Enclose their values in quotes. Field: The name of the attribute, other than the primary attribute. PickField: The name of the business component field. You can define PickMaps to populate as many fields as desired. Number the PickMaps in sequential order, for example Pickmap03, PickMap04, and so on. Define one PickMap for each field.
PickMap03	This PickMap defines the business component field used to filter the records displayed in the pick applet Field, PickField, and Constrain variables are required. Enclose their values in quotes. Field: Specifies the business component name and field that filters the pick applet. The format for specifying the field name is `buscompname.fieldname`. The business component specified in Field must be in the same Tools business object (BusObj) as in Siebel Configurator. The field cannot be the field specified in PickMap01. For example, to constrain the records displayed in the pick applet to those having the current record's Opportunity name in the Quotes view, (Quote business component), you would enter `Quote.Opportunity`. PickField: Specifies the name of field in the picklist that is filtered by the Field variable. Constrain: Must be set to "Y". All PickMaps must be have a unique number. For example if there are 4 PickMaps, PickMap01...PickMap04, name the constraint PickMap, PickMap05.

To use a field to constrain the user's choices

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. Select and lock the desired product with components.
3. Navigate to the Properties view.
4. Select the desired attribute.
5. Define the UI properties as shown in Table 7 on page 93.

Creating an Attribute Value Constraint

You can constrain the records that display in the pick applet based on the value of an attribute in the selection pages. The attribute value is used to create a search specification that matches the attribute value to the value in a field in the business component that populates the pick applet.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

For example, you define an attribute called Account Name. In a configuration session, the user selects Hewlett Packard from the pick applet you have defined for this attribute. You have also created an attribute called Address. You have defined a pickmap for this attribute that constrains the display of addresses to those belonging to the value of the Account Name attribute, which is Hewlett Packard. The pick applet for Address would display only those addresses for Hewlett Packard.

You create an attribute value constraint in the same way as creating a business component constraint. The only difference is that for Field you specify the attribute name rather than a `buscomp.fieldname` in the pickmap definition. Using this example, you enter `Field="Account Name"`.

NOTE: If the attribute you specify to constrain the pick applet does not have an attribute value, the pick applet will contain no records.

Creating a Search Expression Based on the Current Instance

You can constrain the records that display in the pick applet based on an expression that can be computed at run time and is dynamic in nature. The expression can refer to the values in the configuration session. The product administrator can use the standard Siebel Configurator path syntax to define instance variables or values within the search expression.

You do this by defining a PickSpec UI property on an attribute. This additional UI property defines how to constrain the records in the pick applet by specifying the search expression to use to filter the records in the pick applet.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

For example, the Product Administrator has created a product with components called Premier Service Package. This product has been assigned to a product class that has the attributes Job Role, Contact Last Name, Contact First Name defined on it. The Job Role attribute is a drop-down menu with values Please Specify, Manager, and Business Analyst with default attribute value set to Please Specify. The Contact attribute launches the Contact pick applet. These attributes have been added to a group in the User Interface view and will display in selection pages. The Product Administrator wants users to be able to select the Job Role when configuring the product in a quote. When they do, the Product Administrator wants to constrain the Contact pick applet based on the Job Role. For example, if the attribute Job Role is set to Manager and the quote's account name is Boeing, the contact pick applet displays only the contacts who are managers that belong to the Boeing account.

To do this, the Product Administrator must define the following UI properties on the Contact attribute:

- **PickList.** Its value is PickList Contact

- **PickApplet.** Its value is Contact Pick Applet
- **PickMap01.** This UI property provides the name of the attribute and the business component field. Its value is an XML tag that has the following elements:
  - Field = "Contact Last Name". This is the attribute name.
  - PickField = "Last Name". This is the business component field.
- **PickMap02.** This UI property defines an attribute that will receive its value automatically when the user selects a value for the primary attribute. In this case, the value is Contact First Name. Its value is an XML tag that has the following elements:
  - Field = "Contact First Name". This is the attribute name.
  - PickField = "First Name". This is the business component field.
- **PickSpec01.** This UI property filters the display of records in the pick applet based on a search expression. The expression can include references to the current instance. Its value is an expression that has the following elements:

  [PickField] = 'CfgEval('$Cur.xa[Attr Name].xf[Value]')'

  - PickField = "Job Title". This is the business component field.
  - Attr Name = "Job Role". This is the attribute name of the product

  The expression displays all the contacts whose job title is the same as it is displayed in the Job Role attribute.
- **PickSpec02.** This UI property defines additional search specifications. The value is an expression that has the following elements:

  [PickField] = 'CfgEval('$Cur.xa[Attr Name].xf[Value]')'

  - PickField = "Account". This is the business component field.
  - Attr Name = "Account Name". This is the attribute name of the product. The Account Name attribute is bound to the Quote's Account field using the LineItemICField property. For more information, see "Binding an Attribute Value to a Line Item Integration Component Field" on page 74.

  The expression displays all the contacts whose account name is the same as it is displayed in the Account Name attribute.

The Job Role attribute displays with a combo box next to it, and the Contact Last Name and First Name attributes display with a text box next to them in the configuration selection pages. The user sets the Job Role attribute as Manager. When the user clicks the Select button for Contact Last Name, a pick applet displays. It contains only the contacts who are managers that are associated with the account name displayed in the Quote Account field.

Table 8 shows how to use the predefined UI properties to constrain the user's choices.

Table 8. UI Properties

Name	Value
PickSpec 01	This is a search expression used to filter the records displayed in the pick applet. The search expression can include references to the current instance. ■ You can specify search expressions based on the picklist business component fields. The syntax for the search spec expression will be the same as in BC Search Specification in Oracle Web Tools. For example, to create a pick spec for showing records in Contact BC that have Last Name starting with A, set the pickspec to: [Last Name] like "A%". ■ You can specify search expressions based on Attribute names and values. Use the ".xa" operator to return attribute name and the ".xf" operator to return the value. To find the value of an attribute, the CfgPath looks like the following: $.[Root Product].[Rel Name][Product]#InstNum.xa[Attr Name].xf[Value] For example: $.[Bank Prod].[Banking Rel][Checking Prod]#1.xa[Role].xf[Value] ■ You can specify search expressions based on line item fields. Use the ".xf" operator to retrieve the field value. To find the field of a product, the CfgPath looks like the following: $.[Root Product].[Rel Name][Product]#InstNum.xf[Field Name] For example: $.[Bank Prod].[Banking Rel][Checking Prod]#1.xf[List Price] ■ You can specify search expressions using the CfgEval() function. Replace the string CfgEval(…) with the value of the evaluation. To compare strings in Siebel search expression, the CfgPath needs to be enclosed in single quotes. For example, to create a pick spec for showing records in Contact BC whose job title is the same as it is displayed in the Job Role attribute, you can use the following: [Job Title] = 'CfgEval('$.[Bank Prod].[Banking Rel][Checking Prod]#1.xa[Job Role].xf[Value]')' OR [Job Title] = 'CfgEval('$Cur.xa[Job Role].xf[Value]')' OR [Job Title] = 'CfgEval('$Int.[12-BANK34].[Banking Rel][Checking Prod]#1.xa[Job Role].xf[Value]')' OR [Job Title] = 'CfgEval('$Int.[12-CHK34].xa[Job Role].xf[Value]')' ■ You can specify search expressions to support the Integration ID. When using the Integration ID format, the Integration ID is enclosed in the first square brackets. The Integration ID may be used when you want to refer to a particular instance of a product instead of a random instance. All PickSpecs must have a unique number. For example if there are 4 PickSpecs, name the searchspecs PcikSpec01…PickSpec04. **NOTE:** PickSpecs may include If...Else...If...Else statements (i.e IIF expressions). However, CfgEval functionality is not supported within the IIF expression at this point in time.

To use an expression to constrain the user's choices

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. Select and lock the desired product with components.
3. Navigate to the Properties view.
4. Select the desired attribute.
5. Define the UI properties as shown in Table 8 on page 98.

Refreshing Attributes with Business Component Domains

Attributes with business component domains can be refreshed from the database during the configuration session initialization. You can define a new set of join like UI properties on attributes. When the user starts the configuration session, the attribute values get updated from the line item field values. When refreshing the business components returns no rows, then the attribute values are left unchanged.

This task is a step in "Process of Creating an Attribute with a Business Component Domain" on page 87.

For example, the Product Administrator has created a product with components called Premier Service Package. This product has been assigned to a product class that has the attributes Last Name, and Role defined on it. The end user picks a contact using the contact pick applet defined on the Last Name attribute, and the Role attribute is defaulted to Business Analyst. The contact gets promoted to a Manager, and the contact's new job title information is updated in the application. The end user re-customizes the product Premier Service Package. When Siebel Configurator is launched, the job title attribute gets automatically refreshed from Business Analyst to Manager.

To do this, the Product Administrator must define the following UI properties on the Contact Last Name attribute:

- **RefreshBO**. Its value is Contact
- **RefreshBC**. Its value is Contact
- **RefreshKey**. Its value is Id
- **RefreshMap01**. This UI property associates the attribute name with the business component field that needs to be refreshed. Its value is an XML tag that has the following elements

  <RefreshMap Field="Role" RefreshField="Job Title"/>

  - Field = "Role". This is the attribute name.
  - RefreshField = "Job Tile". This is the business component field.

Table 9 shows how to use the predefined UI properties to refresh attributes with business component domains.

Table 9. Predefined UI Properties

Name	Value
RefreshBO	The name of the refresh business object.
RefreshBC	The name of the refresh business component.
RefreshKey	The name of the unique identifier, the Id field of the refresh business component.
RefreshMap 01	This is an XML tag that associates the attribute name with the business component field that needs to be refreshed. <RefreshMap Field="Attribute Name" RefreshField="BuscompFieldName"/> ■ **Field.** The attribute name in the selection page. ■ **RefreshField.** This is the business component field to be used as the attribute value.
RefreshMap nn	RefreshMap02 and so on, refresh multiple fields from the same business component.

8 Smart Part Numbers for Products with Attributes

This chapter explains how to set up part numbers so that they are dynamically generated based on the product attributes that the user selects. Smart part numbers can be used to generate part numbers when creating quotes, orders, and agreements. They can also be used when adding items to a shopping cart.

This chapter includes the following topics:

About Smart Part Numbers

Smart part numbers can be used to generate part numbers for the following types of products in quotes, orders, agreements, and for products added to shopping carts:

- Simple products
- Bundles
- Products with components

NOTE: The Smart Part Number Generation business service generates the Smart Part Number only for a product that has attribute values defined for its product class.

Smart part numbers allow you to automatically generate part numbers for different combinations of product attributes. You do not have to make an entry in the product table and provide a part number for each combination of product attributes that a customer can purchase.

For example, you sell shirts in three sizes: small, medium, and large. You also sell them in three colors: red, green, and blue. There are nine possible combinations of size and color that customers can purchase. Each combination needs a part number that can be passed to a back-end application at the time of purchase.

One way to set this up is to make an entry in the product table for each combination. In other words, you create nine separate products. This is time consuming and does not take advantage of the attribute features in the class system. It also does not take advantage of the attribute-based pricing features in Siebel Pricer.

Another way to set this up is to make one entry in the product table for the shirt. You then define color and size attributes on the class to which the shirt belongs. Finally, you use smart part number to define which part number to assign to each combination of attributes. For example, when the customer selects the shirt in size small and color blue, smart part number generates a part number for this combination and displays it in the quote. You can also use attribute-based pricing in Pricer to determine the price of the shirt.

There are several advantages to this method:

- It makes managing the product table easier. You make one entry for a product and then use the class system to define and manage its attributes. If you enter a product's attribute combinations as products in the product table, you must manually edit the table when attributes change.
- It makes managing part numbers easier and more accurate. You can make one entry for a part number definition and it will be applied to all the forms of the product consistently and accurately.
- It allows you to take advantage of important features in related products such as attribute-based pricing in Siebel Pricer.

Roadmap for Creating Smart Part Numbers

Smart part number provides the following methods for defining how part numbers are generated:

- **Dynamic.** You specify what product attributes participate in creating a part number and the string that each attribute value will have. You then define a part number template with placeholders for the attribute values. Smart part number inserts the value of the attribute into the part number template to create the final part number. Use this method when your part numbers include important information, besides attribute values, that is needed to uniquely identify the product.
- **Predefined.** You specify what product attributes participate in creating a part number. You can then do one of two things:
  - You can auto-generate a matrix of all the combinations of these attributes. Random part numbers are provided for each combination. You can accept the random values or replace them with your own values.
  - You can manually create the matrix, inserting your own part numbers.
  - When the user selects product attributes, smart part number searches the list for the correct attribute combination and uses its part number. Use this method when your part numbers cannot be easily created using string substitution.

You define named smart part number methods on product classes. These methods use the attributes defined on the class to generate a part number. Only attributes with a list of values domain can be used to generate part numbers. When you assign a product to a class, you can select for it any of the smart part number methods that have been defined on the class.

If you add or remove attributes on a class, or change the values for an attribute, these changes are not automatically propagated to the smart part number methods defined on the class. You must manually update each smart part number method with the changes.

To create smart part numbers for products, perform the following tasks:

1. Creating the smart part numbers. You can do this in one of two ways:
   - "Process of Creating Dynamically Generated Smart Part Numbers" on page 103
   - "Process of Creating Predefined Smart Part Numbers" on page 107
2. "Assigning Smart Part Numbers to a Product" on page 110

Process of Creating Dynamically Generated Smart Part Numbers

When you create dynamic part numbers, you first create a template that contains a placeholder for each attribute you want to include in the part number. You then define mappings that specify how attribute values replace the placeholders. When the user chooses attribute values, the application inserts the mappings into the part number template to generate the part number.

The following definitions are important to understanding dynamically generated part numbers.

- **Part number template.** A sequence of sections in a specified order.
- **Section.** A portion of a part number template. Each section contains one attribute name that acts as a placeholder. Sections can also contain a prefix and a postfix.

  For example, you want to create part numbers that begin with ENU- and end with -MC. You want to include values for two attributes, Attrib1 and Attrib2, and separate them with a dash (-). Here is an example: ENU-S-GRN-MC. In this part number, S is the value substituted for Attrib1 and GRN is the value for Attrib2.

  To create a part number template, you would define two sections. In Table 10, the first row is the first section of the part number. The second row is the second section.

Table 10. Part Number Template

Prefix	Attribute Name (Placeholder)	Postfix	Sequence
ENU-	Attrib1	-	1
	Attrib2	-MC	2

- **Mapping.** A mapping is a string of characters you define for an attribute value. The mapping is what the abbreviation method uses to determine what characters to insert in the part number. If you do not define a mapping, the abbreviation method uses the attribute value itself.
- **Abbreviation method.** The abbreviation method determines how the characters are derived from the mapping. These characters replace the attribute's placeholder in the part number template. The mapping methods are: Abbreviation, Acronym, First Two Symbols, First and Last Symbols, First Symbol.
  - **Abbreviation.** Inserts the whole mapping.
  - **Acronym.** Inserts the first character in the mapping plus the first character following each space in the mapping.
  - **First Two Symbols.** Inserts the first two characters in the mapping.
  - **First and Last Symbols.** Inserts the first and last characters in the mapping.
  - **First Symbol.** Inserts the first character in the mapping.

An example of how abbreviation methods determine which characters to insert in part number templates is shown in Table 11. The first column shows the mapping. The remaining columns show the characters that would be inserted in the template for each abbreviation method.

Table 11. How Mapping Methods Work

Mapping	Abbreviation	Acronym	First Symbol	First and Last Symbols
Small	Small	S	S	Sl
X Large	X Large	XL	X	Xe
X-LARGE	X-Large	X	X	XE
A123 B456 C78	A123 B456 C78	ABC	A	A8

Only attributes with a list of values domain can be used to create dynamic part numbers.

Before creating part numbers using this method, determine which attributes you want to use in the part number. Then write down the sections of the part number, including the prefix and postfix for each attribute.

To creating dynamically generated part numbers, perform the following tasks:

1. "Creating a Part Number Generation Record" on page 105
2. "Defining the Part Number Templates" on page 105
3. "Mapping Attribute Values to the Template" on page 106
4. "Testing the Part Number Templates" on page 106

Creating a Part Number Generation Record

First Create a part number generation record. This record names the part number generation method and specifies whether the method is dynamic or predefined.

This task is a step in "Process of Creating Dynamically Generated Smart Part Numbers" on page 103.

To create a part number generation record

1. Navigate to the Administration - Products screen, then the Product Classes view.
2. In the Product Classes list, select the desired product class.
3. Click the name in the Smart Part Number field.

   The name is a hyperlink. The Smart Part Number list appears.
4. In the Smart Part Number list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter a name for the part number generation method.
Type	Select Dynamic.

Defining the Part Number Templates

Next, define the part number template.

This task is a step in "Process of Creating Dynamically Generated Smart Part Numbers" on page 103.

To define the part number template

1. In the Name field of the Smart Part Number record, click the name you entered.

   The name is a hyperlink. The Part Number Method view appears.
2. In the Part Number Template list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Pre-Fix	Enter a static text that appears before all other parts of the smart part number generated by the application
Attribute Name	Select the attribute for this section
Post Fix	Enter a static text that appears after all other parts of the smart part number generated by the application

Field	Comments
Abbreviation Method	Select an abbreviation method for the attribute mapping
Sequence	Enter a sequence. The sequence determines the order of the section in the part number.

3 Repeat Step 2 for each section you want to include in the part number.

Mapping Attribute Values to the Template

Next, map attribute values to the template.

This task is a step in "Process of Creating Dynamically Generated Smart Part Numbers" on page 103.

To define a mapping for each attribute value

1 In the Part Number Template list, highlight the template section for which you want to define attribute value mappings.

The values for the attribute display in the Attribute Mapping list, after the Part Number Template list.

2 For each attribute value in Attribute Mapping, enter a mapping in the Mapping field.

The abbreviation method uses the string in the Mapping field to determine what characters to insert in the part number for this attribute value. If you do not enter a mapping, the abbreviation method uses the attribute value as the mapping.

3 Repeat Step 1 and Step 2 for each section in Part Number Template.

Testing the Part Number Templates

Finally, test the part number template.

This task is a step in "Process of Creating Dynamically Generated Smart Part Numbers" on page 103.

You do this by creating a quote and selecting the product for which you have created a smart part number method. Add the product to the quote enough times so that you can select all the combinations of attributes needed to verify that the smart part number template is working correctly. The smart part number displays in the Line Item Detail view. For more information on locating the smart part number in a quote, see "Viewing a Product's Smart Part Number in a Quote" on page 110.

Editing a Dynamic Generation Method

You can edit a dynamic generation method in several ways:

- Edit the name of the generation method

- Delete the generation method
- Edit the part number template
- Edit the attribute value mappings

If you edit the name of a generation method or delete the method, the change is reflected in all product records to which the method is assigned. For example, you delete the generation method Dynamic1. All product records that have Dynamic1 as the Part Number Method, no longer have an assigned generation method.

If you edit the product template or attribute mappings, the changes become effective immediately. The next time the product is added to quote, order, and so on, the revised part number scheme will be used. The part numbers assigned to products are not changed. You can update the part number by reselecting the product attributes.

If you add or remove attributes defined on a class or change attribute values, these changes are not propagated to the generation method. You must manually update the method. For information on this, see "Updating a Generation Method with Attribute Changes" on page 111.

Process of Creating Predefined Smart Part Numbers

To create predefined part numbers, you create a matrix that contains one row for each possible combination of attribute values. The last entry in the row is the part number you want to assign to this combination. You can create the matrix manually, or the application can generate it automatically.

When the application generates the matrix, it assigns a random part number to each combination. You can accept this part number or replace it with one of your own.

When the user configures the product, the application searches the matrix for the combination of attribute values the user has chosen and assigns the corresponding part number to the product.

Only attributes with a list of values (LOV) domain can be used to create predefined part numbers. Before creating part numbers using this method, determine which combinations of attribute values are allowable. Assign part numbers only to these combinations.

To create predefined part numbers, perform the following tasks:

1. "Creating a Part Number Generation Record" on page 107
2. "Selecting the Attributes for Predefined Part Numbers" on page 108
3. "Creating the Part Number Matrix" on page 108
4. "Testing the Part Number Matrix" on page 109

Creating a Part Number Generation Record

First, create a part number generation record. This record names the part number generation method and specifies whether the method is dynamic or predefined.

This task is a step in "Process of Creating Predefined Smart Part Numbers" on page 107.

1 Navigate to the Administration - Products screen, then the Product Classes view.

2 In the Product Classes list, select the desired product class.

3 Click the link in the Smart Part Number field.

4 In the Part Number Definitions list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter a name for the part number generation method.
Type	Select Predefined.

Selecting the Attributes for Predefined Part Numbers

The next step is to select the desired attributes.

This task is a step in "Process of Creating Predefined Smart Part Numbers" on page 107.

To select the desired attributes

1 In the part number definition Name field, click the name you entered.

The name is a hyperlink. The Part Number Method view appears.

2 In the Attributes list, add a new record and, in the Attribute Name field, select the desired attribute.

3 Repeat Step 2 until you have added all the attributes that you want to use for defining part numbers.

Creating the Part Number Matrix

The last step is to generate a part number matrix.

This task is a step in "Process of Creating Predefined Smart Part Numbers" on page 107.

To create the part number matrix

1 Click the Attribute Matrix view tab.

The Attribute Matrix displays the part number matrix. There is one column for each attribute you selected. There is also a Part Number column and a Description column.

2 To generate the matrix automatically, click the menu button and choose Generate Part Numbers.

The application creates one record for each possible combination of attribute values. The application also generates a random part number for each combination.

NOTE: You can also create the matrix manually by clicking New and creating a record for each desired attribute combination.

3 Review the matrix and verify that it is structured correctly.

If you have not specified the correct attributes, click Attributes. Then add or subtract attributes as needed before regenerating the matrix.

4 Edit the part number for each attribute combination as desired.

You can either accept the randomly generated part numbers or enter the desired part numbers. You can also enter a description for each combination. Users do not see the description.

5 Add new records as desired.

Enter an attribute value for each attribute, and enter a part number.

6 Delete records for unneeded attribute combinations as desired.

Testing the Part Number Matrix

The last step is to test the part number matrix.

This task is a step in "Process of Creating Predefined Smart Part Numbers" on page 107.

You do this by creating a quote and selecting the product for which you have created a smart part number method. See "Viewing a Product's Smart Part Number in a Quote" on page 110.

Editing a Predefined Generation Method

You can edit a predefined generation method in several ways:

- Edit the name of the generation method
- Delete the generation method
- Edit the part numbers in a generation method's part number matrix
- Add or delete records in a generation method's part number matrix
- Regenerate the part number matrix, using different attributes

If you edit the name of a generation method or delete the method, the change is reflected in all product records to which the method is assigned. For example, you delete the generation method Predefined1. All product records that have Predefined1 as the Part Number Method, no longer have an assigned generation method.

If you edit the part number matrix for a generation method, the changes become effective immediately. The next time the product is added to quote, order, and so on, the revised part number scheme will be used. The part numbers assigned to products are not changed. You can update the part number by reselecting the product attributes.

If you add or remove attributes defined on a class or change attribute values, these changes are not propagated to the generation method. You must manually update the method. To do this, see "Updating a Generation Method with Attribute Changes" on page 111.

Assigning Smart Part Numbers to a Product

Smart part numbers defined on a product class are not inherited by the products in the class. You must manually assign the part number generation method to a product.

When you assign a generation method to a product, this method is used for generating part numbers whenever this product is used in new quotes, orders and so on.

To assign a generation method to a product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the desired product.
3. In the final applet, in the product Part Number Method field, enter the generation method information.
4. Click Release to make it take effect.

Viewing a Product's Smart Part Number in a Quote

The part number displayed in the Part # field throughout the application and in quotes, orders, and so on is the internally assigned part number. This part number is different than the smart part number, which displays in a separate field.

Before viewing a product's smart part number in a quote, you must assign a part number generation method to the product. See "Assigning Smart Part Numbers to a Product" on page 110.

Assigning a generation method to a product does not cause a smart number to be generated in existing quotes containing the product.

To view a product's smart part number in a quote

1. Create a quote containing the product.
2. Navigate to the Quotes screen, then the Line Items view.
3. Highlight the desired line item, and click the Item Detail view tab.

4 Locate the Smart Part Number field.

You may need to expand the Line Item Detail form to make the Smart Part Number field visible.

Updating a Generation Method with Attribute Changes

When you add or remove attribute definitions for a class, these changes are not propagated to smart part number methods defined on the class. If you modify the list of values domain for an attribute, these changes also are not propagated. You must manually update each smart part number method with changes to attributes.

You do this by validating the smart part number generation method. When you validate a generation method, the application does two things:

- If you have added or removed attributes, a pop-up message displays and recommends you edit the attribute list you are using for the generation method. For dynamic generation methods, you must modify section definitions and mappings. For predefined methods, you must edit the rows of the matrix.
- If you have changed attribute values for an attribute, the changes are added to the attribute values available for selection. For dynamic generation methods, you must edit the mappings to reflect the new values. For predefined methods, you must edit the rows of the matrix.

Choose one of the following procedures to validate a smart part number generation method.

To update a dynamic generation method with attribute changes

1 Navigate to the Administration - Product screen, then the Product Classes view

2 In the Product Classes list, select the desired product class.

3 Click the hyperlink in the Smart Part Number field of the desired product class.

A list of the part number generation methods defined on the product class appears.

4 To edit a generation method name, click the method name in the Name field.

The Part Number Method view appears.

5 In Part Number Template, click the menu button and choose Validate Definition.

If an attribute has been added, removed, or its name has been changed, a pop-up message appears recommending you revise the sections in Part Number Template.

If an attribute's values have changed, the values available in Attribute Mapping are updated and no pop-up message appears.

6 Revise the sections defined in Part Number Template as needed.

7 Add or revise mappings in Attribute Mapping as needed.

The following procedure shows how to update a predefined generation method.

To update a predefined generation method with attribute changes

1. Navigate to the Administration - Product screen, then the Product Classes view.
2. In the Product Classes list, select the desired product class.
3. Click the Part Number Definitions view tab.

   A list of the part number generation methods defined on the product class appears.
4. To edit a generation method name, click the word Predefined in the Type field.

   The Part Number Method view appears.
5. In Attributes, click the menu button and choose Validate Definition.

   If an attribute has been added, removed, or its name has been changed, a pop-message up appears recommending you revise the Attributes list.

   If an attribute's values have changed, the values available for automatically generating a matrix are updated and no pop-up message appears.
6. Revise the Attributes list as needed.
7. Add, remove, or revise rows in the matrix as needed.

9 Designing Products with Components

This chapter describes how to design the structure of a product with components. It includes the following topics:

About Products with Components

A product with components is one that has components that the user can select. For example, you sell desktop workstations. At the time of purchase, the user can select from several types of disk drive, monitor, keyboard, and mouse to configure the workstation.

A product with components can have other products with components as components. For example, you sell a telephone PBX system that includes 6 rack-mounted PC-based modules. Each module is configurable in a fashion similar to a desktop computer. The components of the PBX system form a product hierarchy. To configure the PBX, the user begins at the start of the hierarchy with the PBX as a whole, and then works down through the hierarchy, configuring its components.

About Products with Components and Product Classes

This chapter and subsequent chapters describe how to define structure, constraints, custom user interface, and other properties of individual products.

Alternatively, you can also define these properties for product classes and then associate individual products with the class. The individual products inherit all the properties from the class.

For more information about product classes, see "About Product Classes" on page 60.

To associate a customizable product with a product class

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the desired customizable product.
3. In the Versions list, in the record for the Work Space version, in the Product Class field, select the product class.

About Relationships

You specify the components of a product with components by defining relationships. A relationship can be defined for a single product, a group of products, or the products in a class.

NOTE: Relationships are also called ports.

The relationships you define for a product with components are component type relationships. This means the items in the relationship are components of the product with components. For example, you define a relationship called Hard Drives for the product with components Desktop Computer. You specify that it contains all the products assigned to the Disk Drive class. This makes the disk drives in this class components of the product with components Desktop Computer.

Relationships form the framework of a product with components. They are also the framework underlying the user interface you design for the product. For example, you sell configurable computers. The buyer can choose among several monitors, several keyboards, and several CD-ROMs when configuring a computer. You could create a relationship called Monitors, another called Keyboards, and one called CD-ROMS. You would then specify the products to include in each relationship. You could then design the user interface to present monitors, keyboards, and CD-ROMs each on a separate selection page.

When you design a product with components, begin by defining a framework of relationships. Keep in mind that each relationship represents a distinct, configurable part of the product.

Figure 3 shows a relationship framework in a product with components.

- Relationship 1 contains a single product, Product 1
- Relationship 2 contains all the products in product class, Class 1

- Relationship 3 contains Product 2, Product 3, and Product 4, each from a different product class

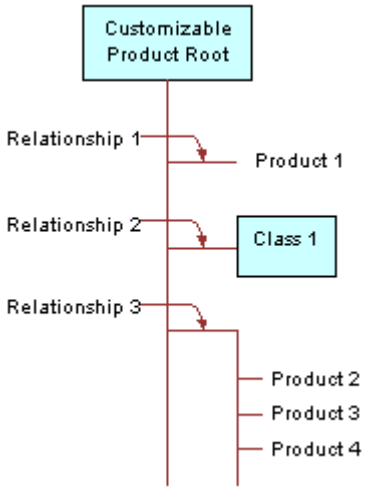

Figure 3. Product with Components Relationships

Products with components and the product class system both include hierarchies. However, these hierarchies differ in important ways. In the product class system, inheritance is used to propagate attribute definitions downward through the class system. By contrast, inheritance plays no role in the hierarchy of components in a product with components. Attributes inherited by a product with components because of its membership in a product class do not propagate to the component products in the product with components.

For example, a product with components belongs to a product class that has the attribute Color (red, green, blue). The product with components as a whole inherits this attribute but its components do not. For example, if the product with components is a laptop computer, this means the laptop comes in three colors, red, green, or blue.

However, these colors are not inherited by any of the components of the laptop. For example, if the laptop has a CD-ROM, it does not inherit these colors. The color attribute of the CD-ROM (if it has one) is defined in the product class from which it comes, not in the product with components in which it resides.

If a class relationship is assigned to a class that have subclasses, the products in the subclasses appear in the class relationship. For example, there is a parent class called Hard Drive and subclasses called Laptop Hard Drives and Desktop Hard Drives, both of which have products in them. If the relationship is on the Hard Drive class, the application will pick up products from both Laptop Hard Drive class and Desktop Hard Drives.

About Cardinality

When you define a relationship, you can specify a minimum, maximum, and default cardinality. Cardinality refers to the quantity of the component that the user can select. For example, you define a relationship called Hard Drives. It contains a 20 GB drive and a 30 GB drive. If you set the minimum cardinality to 2, the user must pick 2 items from this relationship. The user can do this in any of the following ways:

- Pick one 20 GB drive and one 30 GB drive
- Pick two 20 GB drives
- Pick two 30 GB drives

The three types of cardinality you can define for a relationship are as follows:

- **Minimum Cardinality.** Governs whether or not selecting items from this relationship is optional or is required. If you set the minimum cardinality to 0, selecting items is optional. If you set the minimum cardinality to greater than 0, the user must select that number of items from the relationship.
- **Maximum Cardinality.** Sets the maximum number of items that the user can select from a relationship. If you set the minimum cardinality to greater than 0, you must set the maximum cardinality to a number at least as large If you do not enter a maximum cardinality, the default is 999. To revise this default, see "Revising the Default Cardinalities" on page 412.
- **Default Cardinality.** Specifies what quantity of the default product is automatically added to the initial solution that the user sees. Default cardinality must be equal to or greater than the minimum cardinality and must be less than or equal to the maximum cardinality.

  If you specify a default cardinality and do not specify a default product, the application uses the first product that displays when you expand the relationship folder in the Structure view.

Combinations for Setting Cardinality

Table 12 describes several combinations for setting cardinality. The table shows what the user will see in the initial solution and what actions that the user can take. In the table, N is the quantity of the default product in the initial solution. In all the cases where the Min Card is greater than 0, the user can substitute other products for the default product.

Table 12. Combinations of Cardinality

Min Card	Default Card	Max Card	Application Adds Default Product?	User Pick Reqd?	Initial Solution	User Actions Allowed
=0	= Min Card	>Default Card	No	No	No items	Increase item quantities to Max Card.
=0	> Min Card	=Default Card	No	No	N=Max Card	Decrease Item quantities to 0 but cannot increase them.

Table 12. Combinations of Cardinality

Min Card	Default Card	Max Card	Application Adds Default Product?	User Pick Reqd?	Initial Solution	User Actions Allowed
=0	> Min Card	>Default Card	No	No	N=Default Card	Increase item quantities to Max Card or decrease them to 0.
>0	= Min Card	=Default Card	Yes	Yes	N=Min, Default, Max	Cannot increase or decrease item quantities.
>0	= Min Card	> Default Card	Yes	Yes	N=Min	Can increase item quantities to Max Card but cannot decrease them.
>0	> Min Card	= Default Card	Yes	Yes	N=Default	Can decrease item quantities to Min Card but cannot increase them.
>0	> Min Card	>Default Card	Yes	Yes	N=Default	Can decrease item quantities to Min Card or increase them to Max Card.

About Generics

Generics are notifications to the user from the engine that one or more items within a relationship needs to be selected for the product with components to be correct. An example of generics is a star displayed next to the relationship name and product title during a configuration session because some minimum cardinality requirements were not satisfied. Siebel Configurator gives users a warning when they try to save a configuration that has generics in it, but it allows users to save the configuration and also saves the fact that the configuration is incomplete to the quote or order.

When the user verifies a quote, the application checks for incomplete configurations as well as for other information. If the configuration saved was incomplete because of unsatisfied cardinalities on relationships, the application displays a message when the user verifies saying that the configuration of the item is not complete and the user must reconfigure the item.

Guidelines for Designing Products with Components

You can minimize order problems if you avoid adding the same product to more than one relationship in a customizable product. If you restructure the customizable product and publish a new version, the Auto Match business service in Siebel Configurator may not pick the correct relationship for the item if transactional data (for quotes, orders, and assets) based on old versions of the product is reconfigured. For more information see

"Auto Match Business Service for Siebel Configurator" on page 417.

CAUTION: Be sure that the configurable products have a valid solution for all initial conditions. It is possible to create a configurable product that does not have a valid solution for certain initial conditions, so that this product cannot be instantiated at all. For example, this problem may occur when a set of hard constraints, based on the linked items, is violated. However, this will lead to an internal error in the Siebel Configurator engine.

Guidelines for Asset-Based Ordering

If you are designing products with components and you use asset-based ordering, the best practice is to avoid creating require rules that add items that are not tracked as assets to a customizable product.

For example, you write a require rule that adds a one-time charge for Installation to a customizable product. You do not set the Track as Asset flag for Installation in its product record. This means Installation does not display as a customer asset.

Then the customer requests an addition to the service. The call center agent selects the service, clicks Modify, and starts a configuration session. The Siebel Configurator engine adds Installation, because it is required by configuration rules. Installation is transferred to the quote even though it is not required by the service modification.

Process of Designing a Product with Components

To define a product with components, perform the following tasks:

- "Creating Product Records for a Product with Components and for Its Components" on page 119
- Add components to products with components, which can be done in the following ways:
  - "Adding a Single Product as a Component" on page 119
  - "Adding Products as Components Using the Class Domain" on page 120
  - "Adding Products as Components Using the Dynamic Class Domain" on page 121
  - "Adding a Group of Products from Different Classes as Components" on page 122
  - "Adding a Product with Components as a Component" on page 124

NOTE: Instead of defining the structure of an individual product with components, as described in this process, you can define the structure of a product class. Products associated with the class inherit both the structure and constraints from the class. For more information, see "About Product Classes" on page 60.

Creating Product Records for a Product with Components and for Its Components

First, you must create product records that represent the product with components as a whole and that represent all of its components.

This task is a step in "Process of Designing a Product with Components" on page 118.

For example, if the product with components is a computer, first you create a product record that for the entire computer and product records for all of its components, such as disk drives, a monitor, and so on. You must create all the products before you can add the components to the product with components.

Create product records in the same way that you do for a simple product, as described in "Process of Creating Simple Products" on page 32.

All these products must be orderable. To make a product orderable, place a check mark in the Orderable check box in the product record.

Adding a Single Product as a Component

You can create a relationship that adds a single product as a component of a product with components.

This task is a step in "Process of Designing a Product with Components" on page 118.

To add a single-product relationship

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. In the Structure list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Relationship Name	Enter a name for this relationship.
Product	Select the product that is a component of this product with components.
Minimum Cardinality, Maximum Cardinality, and Default Cardinality	Enter the minimum, maximum, and default cardinality for the product. For more information, see "About Cardinality" on page 116.

Adding Products as Components Using the Class Domain

You can add products as components of a product with components by using the class domain.

This task is a step in "Process of Designing a Product with Components" on page 118.

This method of adding products does not maintain a connection to the class system. When you refresh the customizable product work space, relationships are not updated. For example, if you assign a new product to a class, this product is not added to the relationship containing this class when you refresh the work space or release a new version of the customizable product. Use this method when you want to keep the relationship contents static or when you want to add only some of the products in a product class. If you want the relationship to be updated when you update the class system, see "Adding Products as Components Using the Dynamic Class Domain" on page 121.

NOTE: Adding a small number of products to a relationship from a large product class requires that the entire class be searched each time the product with components is instantiated. This can adversely affect performance. Consider defining a separate class for these products to avoid this impact on performance.

To add products by using the class domain

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. In the Structure list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Relationship Name	Enter a name for this relationship.
Domain Type	Select Class.
Product Class	Select a class. The dialog box contains one record for each class and for each subclass in the class system. Selecting a class selects all of its subclasses.
Minimum Cardinality, Maximum Cardinality, and Default Cardinality	Enter the minimum, maximum, and default cardinality for the product. For more information, see "About Cardinality" on page 116.

5. Click the Define Domain button and then use the following options to select the products you want to add to the relationship:
   - **Add.** Select products and click Add to add the products to the relationship. A check mark displays in the "Is in domain" field.
   - **Add All button.** Adds all the products in the class to the relationship.

- **Set as Default button.** Adds the product to the relationship and makes it the default product. The product name displays in the Default Product field of the relationship.
- **Clear Default button.** Removes the product from the relationship's Default Product field. Does not remove the product from the relationship.
- **Delete button.** Removes the product from those you have selected to be in the relationship. Removes the check mark from the "Is in domain" field. Does not remove the product from the product class.
- **Delete All button.** Removes all the products from the relationship. No products display a check mark in the "Is in domain" field. Does not remove the products from the product class.

6 In the Product Structure Designer panel, the relationship icon displays as a folder. Click this folder to display the products you added, and verify that the relationship is defined properly, that the default product is correct, and that all the products you want to add are present.

7 In the Relationship Domain list, do the following:

a Remove the check mark from the Forecastable field for items as needed.

Removing the check mark means the item will not be included in product forecasts when the opportunity is updated for quotes, orders, and so on contained the product with components.

b For each product in the relationship, enter a sequence number in the Sequence Number Field.

The item with sequence number 1 displays first within the relationship in selection pages.

Adding Products as Components Using the Dynamic Class Domain

You can add products as components of a product with components by using the dynamic class domain.

This task is a step in "Process of Designing a Product with Components" on page 118.

This method of adding products maintains a connection to the class system. When the work space is refreshed, Dynamic Class relationships are updated from the class system. For example, if you add a new product to a class in the class system, this product is added to the relationship containing this class when you select Refresh Dynamic Class in the Structure tab menu.

When you refresh the work space to update the contents of the relationship, you must reenter the sequence numbers in the relationship definition.

To add products using the dynamic class domain

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Products list, select and lock the desired product with components.

3 In the Versions list, click the Work Space version.

4 In the Structure list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Relationship Name	Enter a name for this relationship.
Domain Type	Select Dynamic Class.
Product Class	Select a class. The dialog box contains one record for each class and for each subclass in the class system. Selecting a class selects all of its subclasses.
Minimum Cardinality, Maximum Cardinality, and Default Cardinality	Enter the minimum, maximum, and default cardinality for the product. For more information, see "About Cardinality" on page 116.

5 Click the Define Domain button and select the products you want to add to the relationship:

a In the Define Relationship Domain dialog box, click Add All.

A check mark appears in the *Is in domain* field for all the products in the class.

b Also, in the Define Relationship domain dialog box, use the following buttons to specify the default product:

- **Set as Default button.** Adds the product to the relationship and makes it the default product. The product name displays in the Default Product field of the relationship.
- **Clear Default button.** Removes the product from the relationship's Default Product field. Does not remove the product from the relationship.

6 In the Relationship Domain list, do the following:

a Remove the check mark from the Forecastable field for items as needed.

Removing the check mark means the item will not be included in product forecasts when the opportunity is updated for quotes, orders, and so on contained the product with components.

b For each product in the relationship, enter a sequence number in the Sequence Number Field.

The item with sequence number 1 displays first within the relationship in selection pages.

Adding a Group of Products from Different Classes as Components

The products you add to a relationship do not have to be from the same class. You can group products from several classes or products not assigned to a class into one relationship.

This task is a step in "Process of Designing a Product with Components" on page 118.

You do this by creating a relationship of domain type Class but without specifying a class. This allows you to select products from anywhere in the class system.

You can do anything with this relationship that you can do with other class-type relationships such as creating resources, configuration rules, and links.

CAUTION: This method of defining a relationship domain requires a search throughout the class system each time the product with components is instantiated. This can have an adverse impact on performance. Avoid using this method, if possible.

To add groups of products from different classes

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. In the Structure list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Relationship Name	Enter a name for this relationship.
Domain Type	Select Class.
Minimum Cardinality, Maximum Cardinality, and Default Cardinality	Enter the minimum, maximum, and default cardinality for the product. For more information, see "About Cardinality" on page 116.

5. Click the Define Domain button and select the products you want to add to the relationship, using the following buttons in the Define Relationship Domain dialog box:
   - **Add column.** Click the word Add in the record to add the product to the relationship. A check mark displays in the "Is in domain" field.
   - **Add All button.** Adds all the products in the class to the relationship.
   - **Set as Default button.** Adds the product to the relationship and makes it the default product. The product name displays in the Default Product field of the relationship.
   - **Clear Default button.** Removes the product from the relationship's Default Product field. Does not remove the product from the relationship.
   - **Delete button.** Removes the product from those you have selected to be in the relationship. Removes the check mark from the "Is in domain" field. Does not remove the product from the product class.
   - **Delete All button.** Removes all the products from the relationship. No products display a check mark in the "Is in domain" field. Does not remove the products from the product class.

   Because you did not specify the product class, this dialog box lists all products from all classes and subclasses.

Adding a Product with Components as a Component

You can add products with components as components of other products with components. This means you can create products with components that are sub-assemblies and then include them as components in the final product. For example, you sell a configurable power supply and a configurable gearbox as part of an industrial lathe. You can create one product with components for configuring the power supply and one for configuring the gearbox. You can then add both of these component products with components to the industrial lathe product with components.

This task is a step in "Process of Designing a Product with Components" on page 118.

In the Structure view, when you add a product with components to a relationship, its configurable parts do not display. Instead, the product with components displays as a single product.

When you edit a product with components and release it, the changes propagate to all products with components containing it.

Use the following procedures to add a product with components:

- To add a product with components that is not assigned to a class, see "Adding a Single Product as a Component" on page 119.
- To add a product with components that is a member of a class, see "Adding Products as Components Using the Class Domain" on page 120 or "Adding Products as Components Using the Dynamic Class Domain" on page 121.

Refreshing the Customizable Product Work Space

If a product with components contains relationships of type Dynamic Class, refreshing the work space copies a new instance of these product classes into the relationships. This means a fresh copy of all the products in the class become part of the product with components instance.

For example, the number of products in a class has changed. You have defined a relationship of type Dynamic Class that specifies this product class. When you refresh the work space, the revised product class is copied to the relationship from the product table. When you view the relationship in the User Interface view or in Validate mode, the new products display.

Relationships of domain type Class and Product are not updated from the product table when you refresh the work space.

Refreshing the work space updates the products or attributes in a product with components. The configuration rules, resource definitions, link definitions, and scripts that are part of the product with components are not updated to reflect changes. You must manually make these updates.

For more information about relationships of type Dynamic Class, see "Adding Products as Components Using the Dynamic Class Domain" on page 121.

NOTE: To use this procedure, the Structure Type field of the product must be set to Customizable.

To refresh the work space

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. From the Structure menu, select Refresh Dynamic Class.

Enabling the Customize Button

The user can select a configurable product and click the Customize button to customize the product. However, the Customize button is only enabled when the user's responsibilities include one of the appropriate views that allows users to customize products. The following views of the Quotes screen are examples of views that enable the customize button:

- Complex Product Runtime Instance View
- Complex Product Runtime Instance View - Order
- Complex Product Runtime Instance View - Shopping Cart

Within these views, the Customize button is enabled only when the type of the product is Customizable.

To find a complete list of the views that enable the Customize button

1. Open Oracle's Web Tools.
2. In the Views tab, search for views that match the following:

   Complex Product Runtime Instance View*

About the Siebel Configurator Save Button

When Asset Based Ordering (ABO) is enabled, the Save button is disabled. If the Save button is not needed, because the application is using ABO, you can remove it by configuring the product using Oracle's Web Tools.

The Save button is similar to the Done button, except that it does not exit from the configuration session. In general, after you click the Save button, clicking Cancel does not remove the items from the configured product.

About Managing the Structure of Products with Components

You can manage the structure of products with components in the following ways:

Editing a Relationship Definition

You can only edit the current work space of a product with components. You cannot edit a version that has already been released. All the fields in a relationship definition can be edited except the relationship name. Changes are not propagated to other parts of the product with components.

To edit a relationship definition

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product with components.
3. In the Versions list, click the Work Space version.
4. In the Explorer applet, select the desired relationship.
5. Edit the fields in the Structure record as desired.
6. Revise configuration rules, resource definitions, link definitions, and scripts as needed to reflect the changes.

Deleting Products from Products with Components

You can only delete products from the current work space of a customizable product. You cannot delete products from a released version. You can delete relationships or products included within a relationship.

Changes are not propagated to other parts of the product with components. For example, if you delete a product from a relationship, configuration rules for that product are not deleted.

If you delete a product from a relationship that has domain type Dynamic Class, the product will be added back to the relationship when you refresh the work space or release the product. This is because the product still exists in the product class. When you refresh the work space or release the product, the relationship is updated so that it contains all the products in the product class and the current attribute definitions.

To avoid this, you can change the relationship domain type to Class. This breaks the connection to the product class system and prevents any further updates of the relationship. You can also leave the domain type unchanged and remove the product from the product class.

To delete products

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product with components.

3 In the Versions list, click the Work Space version.

4 To delete a relationship, select the desired relationship record. To delete a product within relationship, expand the relationship and select the product record.

5 From the Structure menu, choose Delete Record.

6 Revise configuration rules, link definitions, and resource definitions as needed.

Copying Products with Components

When you copy a product with components, all parts of the product are included in the copy. This includes its relationships, links, resources, scripts, rules, and its user interface.

All the parts of the copied product are visible and can be edited.

To copy a product with components

1 Navigate to the Administration - Product screen, then the Products view.

2 In the Products list, select the desired product with components.

3 From the Products menu, choose Copy Record.

4 Enter a name for the copy in the Product Field.

5 Revise other fields, such as Part # as desired.

Creating a Report on a Product's Structure

You can request a report that lists all the relationships in a product with components as well as the contents of each relationship. You can request the report once, or schedule the report to run at scheduled times. The report title is Product Relationship Report. This report must be enabled on the report server before performing the following procedure.

To create a report on a product's structure

1 Remove the product with components from all existing quotes.

2 Navigate to the Administration - Product screen, then the Product Definitions view.

3 In the Products list, select and lock the desired product with components.

4 In the Versions list, click the desired version.

5 Click the Reports icon to produce or schedule the report.

10 Managing Products with Components

This chapter provides information about special techniques for managing products with components. It includes the following topics:

About Auto Match

If a quote, asset, or order contains a product with components configuration based on an out-of-date version of the product, Auto Match can compare the old version with the current version of the product and make limited changes to bring the quote, asset, or order up to date automatically. The user does not have to configure the product again.

Auto Match works as follows:

- Auto Match is triggered when the system determines that the version in the quote, order, or asset is not the current version.
- Auto Match compares the relationships and their contents in the quote, asset or order to the current version of the product.
- Auto Match identifies items in the old version that are not in the same relationship as items in the new version. An item can be a product or product class. These items in the old version are misclassified items. The relationship containing them is the old relationship. The relationship in the current product that contains the items is the new relationship.
- If the old relationship and the new relationship have a common parent, typically the product root, Auto Match will automatically move the items to the new relationship in the quote, order or asset. Auto Match does this by either changing the relationship name or by adding a new relationship.
- If the old relationship and new relationship do not have the same parent, the user receives and error message and must configure the product again.

- If the current version has a lower max cardinality for a relationship, this cardinality is enforced in the version in the quote, order, or asset. For example, the cardinality of Relationship A has been reduced from 10 to 8 in the current version. In a quote, Relationship A contains 10 items. Auto Match will remove two items from the quote.
- If a relationship has been removed from the current version but is included in a quote, order, or asset, Auto Match will attempt to move its items to a relationship at the same level. For example, Relationship A, containing 10 items, has been removed from the current version. A quote has the previous version of the product with components, including Relationship A with 10 items. Auto Match will try to move all 10 items to other relationships at the same level, while observing maximum cardinality restrictions. Any excess items are removed.
- Auto Match only compares the physical structure of the product's current version to that in the quote, asset, or order. It does not consider configuration rules. For example, if the old version contains Product A, and Product A would be excluded in the new version, Auto Match does not detect this.
- For products with attributes, Auto Match checks to see whether any of the required attributes have not been specified.

Auto Match is implemented as a business service and is not enabled by default. To turn Auto Match on, see "Enabling Auto Match" on page 408.

About Finish It!

"Finish It!" is a button that appears in configuration session selection pages. This button is active under the following conditions:

- The configuration session contains a relationship that has a minimum cardinality greater than zero.
- No default product has been defined for this relationship.
- The user has not selected the number of products from this relationship required by the minimum cardinality.

These relationships are called unsatisfied quantity relationships. In selection pages, an asterisk displays next to the relationship name and next to the item in the relationship, indicating that the user must make a selection.

When the user clicks Finish It!, the Siebel Configurator engine adds items to the solution from all unsatisfied quantity relationships so that minimum cardinalities are met or exceeded. the Siebel Configurator engine makes arbitrary selections from these relationships. You cannot specify which products will be selected by setting the sequence of the product in the relationship.

For example, you have defined a relationship called Keyboard. This relationship has a minimum cardinality of 1 and no default product. This causes the Finish It! button to become active in configuration sessions. When the user clicks Finish It!, the Siebel Configurator engine adds a keyboard to the solution.

If you do not want the Finish It! button to be active during configuration sessions, specify default products for all relationships with a minimum cardinality greater than zero.

NOTE: If there are required attributes that have not been selected, Finish It displays a message telling the user those attributes are required. It does not fill in a value for the required attributes. For more information about required attributes, see "Setting Up Required Attributes" on page 70.

Viewing Relationships for Products

Use the Relationships view to view all the relationships associated with a product, so you can see which customizable products use this product as a component.

To view relationships for a product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the product whose relationships you want to view.
3. Click the name of the desired version.
4. Navigate to the Cross References view.

Using Product Classes as Templates for Products with Components

You can create a product class and use it as a template for building other products with components.

Use this feature when you have products with components that include the same group of items. For example, you sell desktop computers. You have seven configurable models that share the same chassis types, keyboards, and mouse. You can create a product class consisting of these three relationships. You would then use the product class as the basis for constructing each model.

The following features of the product class are inherited by all class members:

- Relationships and their contents.
- Configuration rules.
- Resources
- Links
- User interface groups.
- The base theme and product theme are not inherited
- User interface property definitions and scripts are not inherited by class members

To use a product class as a template for a product with components

1. Navigate to the Administration - Product screen, then the Product Classes view.

2 In the Products Classes list, select the product class you want to use as a template.

3 Navigate to the Administration - Product screen, then the Product Definitions view.

4 Lock the product with components.

5 In the Product record for the desired product, in the Product Class field, select the desired product class.

   Adding a product to a product class will be inherited by any customizable product that is associated to the Product Class

6 Navigate to the User Interface view and verify the class-product structure has been inherited correctly.

   NOTE: The user interface defined on product class is automatically inherited to the product class structure. You need to adhere to the following rules if you want to add an additional group at the product or child class level to the inherited user interface:

   a The user interface records must have the same name in each product class hierarchy.

   b Group sequence numbers must be unique and ordered as required in a product class hierarchy.

About Bundles as Products with Components

A bundle is a group of items sold as one product and is a special form of product with components that has the following characteristics:

- Bundles created in Bundle Administration also display in Customizable Products, then Versions and in Customizable Products, then Structure.
- A bundle is made up of one or more relationships that have a Product domain. Each relationship adds only one product to a bundle.
- Bundles do not include a UI definition, configuration rules, links, resources, or scripts. Bundles do not include selection pages. Users do not configure a bundle by starting a configuration session.
- The quantity of a product in a bundle is determined by the Default Cardinality of the product.

If you add items to a bundle that are not allowed, they are ignored. For example, if you define configuration rules for a bundle, they are ignored. When you convert bundles to regular products with components, ignored items then become effective.

If you convert a product with components to a bundle, only the items within the scope of a bundle are used. All other items, such as configuration rules, are ignored.

Regular products with components are added to quotes using Siebel EAI. Bundles are added to quotes using internal code.

For more information about bundle products, see Chapter 5, “Product Bundles.”

For information about converting bundles, see:

- “Converting a Bundle to a Regular Product with Components” on page 133

■ "Converting a Regular Product with Components to a Bundle" on page 133.

Converting a Bundle to a Regular Product with Components

A bundle is a group of items sold as one product and is a special form of product with components. A bundle has one or more relationships that have a Product domain. Bundles do not include a UI definition, configuration rules, links, resources, or scripts.

When you convert a bundle to a regular product with components, you can work with the newly converted product with components in the same way as a regular product with components.

To convert a bundle to a regular product with components

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired bundle product.
3. Click Release.

   This converts the bundle to a regular product with components. A check mark displays in the Customizable Product check box in the Product record. A check mark no longer displays in the Bundle check box.
4. Revise existing quotes and orders as needed to reflect the change.

Converting a Regular Product with Components to a Bundle

When you convert a regular product with components to a bundle, any items outside the scope of a bundle, such as configuration rules, stop being effective and are ignored. They are not erased from the definition and become effective again if you convert the bundle back to a product with components.

Converting has the following effects:

- All previous versions still display in the Administration - Product screen, then the Product Definitions view.
- The current work space is retained, but its contents are altered as described in the following items.
- For relationships that have a Product domain, the product is added to the bundle.
- For relationships that have a Class or Dynamic Class domain, only the product specified in the Default Product field is added to the bundle. If no product is specified, no product is added, even if the Default Cardinality is greater than one.

- The quantity in the Default Cardinality field is used to determine the quantity of the product in the bundle. Other cardinality fields are ignored. If the default cardinality is blank or zero, the quantity of the product in the bundle is blank. When creating a quote, only those products with a quantity greater than or equal to one are displayed in the quote.
- All selection page definitions, UI Property definitions, configuration rules, link definitions, resource definitions, and scripts are ignored and do not become part of the bundle. If you convert the bundle back to a regular product with components, these again become effective.
- When you convert a product with components that inherits part of its structure from a class-product, none of the inherited structure becomes part of the bundle.
- If the Forecastable flag is set for a product in the Structure view and the product becomes part of a bundle, the Forecastable flag remains set for the product in the bundle.

When a user adds a bundle to a quote or order, the bundle and its components display as separate line items. Users cannot start a configuration session.

The product with components must have at least one released version before you can convert it to a bundle.

To convert a regular product with components to a bundle

1. Navigate to the Administration - Product screen, then the Products view.
2. Select and lock the desired product with components.
3. Make any desired changes and release a new version.
4. Navigate to the Bundle Product view, and review the displayed contents.

   The Bundle Product view displays the products in the product with components that will be included in the bundle. If the contents are not correct, revise the product with components and release a new version.
5. In the Bundle Product view, click Modify, and then click Done.

   This converts the product with components to a bundle and releases a new version. A check mark displays in the Bundle field in the product form. The check mark in the Customizable Product field is removed.
6. Revise existing quotes and orders as needed to reflect the change.

Defining an Asset with Components

When creating a quote, users can choose to add to or modify products with components the customer has already purchased. The customer's already-purchased products with components are called assets with components.

The Quotes user creates a quote and selects a customer's asset with components. The user then starts a configuration session and modifies the asset by adding or removing items. The user then saves the changes to the quote. The asset with components, with revisions, displays in the Quotes screen, then the Line Items view.

In Quotes, Line Items, and then Line Item Detail, each item in the asset with components displays a status in the Delta Status field:

- **Existing.** All items that were not changed. These items display no price.
- **New.** All items that have been added or for which the quantity increased. If the quantity of an item increased, the original quantity is shown with status Existing. A second entry in the quote shows the increase, has status New, and displays a price.
- **Removed.** All items that were removed or decreased in quantity. If the quantity of an item is reduced, the new quantity is displayed with the status Existing. A second entry in the quote shows the reduction, and has status Removed. These items display no price.
- **Modified.** All items for which attribute settings have changed. These items display a price.

Delta Quotes differ from Favorites in that only new or changed items from the configuration session have a price. Unchanged items are listed at zero price. When you add a Favorite to a quote, all the items in the product with components have a price. Favorites are new products being sold for the first time. Delta Quote products are items being sold as add-ons or replacement components for products you have already sold.

The Siebel Configurator engine uses the name of the asset with component's product root to determine what product with components to load for the quote configuration session. If a new version of the product with components has been released, the new version is used to modify the asset with components. Any configuration conflicts that result are displayed during the configuration session and must be resolved. Item pricing is not maintained during the configuration session and must be ignored. Pricing is computed when the user saves the configuration to the quote.

When you define an asset with components, it is added to the Customizable Asset dialog box in Quotes, so users can select it when creating delta quotes.

Defining an asset with components requires two steps:

1. Create a customizable asset record.
2. Configure the customizable asset.

To create a customizable asset record

1. Navigate to the Assets screen, then the List view.

2 In the Assets list, add a new record and complete the necessary fields. Some fields are described in the following table.

CAUTION: If you do not assign an account name, the asset with components displays every time the Customizable Asset dialog box is opened, regardless of the account.

Field	Comments
Product	Select the product with components on which the customer's asset with components is based.
Account	Select the customer's account. This field filters the records displayed in the Customizable Asset dialog box in Quotes. The dialog box displays only the assets with components that have the same account name as the account name in the Quotes record.
Asset Description	Enter a descriptive phrase or name that is meaningful to users creating quotes. This field displays in the Customizable Asset dialog box in quotes

To configure an asset with components

1 Review the product with components configuration that the customer has purchased.

Determine if a new version of the product with components has been released and what effect this will have on configuring an asset with components.

2 Navigate to the Assets screen, then the List view.

3 Drill down on the record of the asset with components which you want to configure.

4 In the asset form, click Customize.

This starts a configuration session, similar to those users see when configuring products with components in Quotes. The session displays the selection pages for the product with components that the user purchased.

5 Configure the product with components to reflect what the customer purchased and exit the session. This includes configuring component attributes.

This creates the configured asset with components.

6 In the Assets list, verify that the desired asset with components is highlighted.

7 Navigate to the Attribute view and set the value of attributes defined for the asset with components as a whole.

Controlling How Products with Components Are Taxed

The components of products with components can have different tax rates.

For example, a company may sell a product with components called Concrete Services. As its components, this product may have the cost of concrete, the cost of using a truck to pour concrete, the cost of labor, and the cost of engineering services. In some jurisdictions, these components may be taxed at different rates.

You can control how tax is computed by setting the Tax Subcomponent flag. To tax the components individually, set the Tax Subcomponent flag on the root of the product with components or bundle. If you do not set the Tax Subcomponent flag, the tax is computed on the total price of the product with components or bundle.

If one of the components is itself a product with components, you can set the Tax Subcomponent flag on the component. This causes the tax for that component to be the sum of the tax computations on the components of that component.

Note the following points about taxing components:

- You cannot use a base price for the parent product if the components are taxed individually. The parent product price must be the sum of the prices of all the component products. You will get inaccurate results if you give the parent product a price, and then do delta pricing on components and compute tax at the subcomponent level.
- When you set the Tax Subcomponent flag for the parent product, you can either set this flag or not set it for each component product that has subcomponents. If you do not set this flag on a component, the tax will be calculated for the component product. If you do set the flag on a component, the tax for the component will be based on the tax of its components.
- When you set the Tax Subcomponent flag for a component of a product, you must set this flag for the parent product as well. If you do not, you will tax the entire product at the parent level, and you will also tax the component product. This double counting will cause an inaccurate tax calculation.

The tax will be calculated accurately if you apply volume discounts or bundling discounts to the product. These discounts are applied at the component level, so the tax on each subcomponent will be adjusted to reflect the discount.

CAUTION: You cannot use simple bundles for pricing when tax rates of components are different.

To tax the components of a product with components or bundle

1. Navigate to the Administration - Product screen, then the Products view.
2. In the Products list, select the desired product with components or bundle.
3. Expand the product form and put a check mark in the Tax Subcomponent Flag check box.

Controlling How Products with Components are Forecast

When you add a component to a product with components, a check mark displays in the Forecastable field. This means the component is added to product forecasts when the product with components is included in a quote and the user updates the related opportunity.

To prevent components from being added to product forecasts, remove the check mark from the component's Forecastable field.

A Forecastable check box is also available in Quotes, then Line Items. This allows you to add or remove a product or component from product forecasts within individual quotes.

11 Creating Custom Siebel Configurator User Interfaces

This chapter describes how to create a custom user interface for configuring a customizable product and how to assign user interfaces to users. You can create this interface for products with components and products with attributes.

This chapter includes the following topics:

About Default and Custom Siebel Configurator User Interfaces

You must decide whether to use the default Siebel Configurator user interface or to design a custom user interface.

The default user interface displays all the configurable items in the customizable product on a single selection page. If you create a custom interface, you can display the configurable items on multiple pages.

The default user interface makes intelligent choices for UI controls based on attribute domain type and upon relationship cardinality. For example:

- If an attribute has an LOV type domain, the default interface displays a combo box for setting the attribute.
- If a relationship has a min and max cardinality of 1, the application displays a combo box without multiple instances. If the max cardinality is greater than 1, the application displays a combo box with multiple instances.

If you create a custom interface, you can select the control used for each configurable item, and you can create several Open UI interfaces and assign them to users based on their responsibilities and their browsers.

The application decides which interface to assign to a user using the decision flow shown in Figure 4.

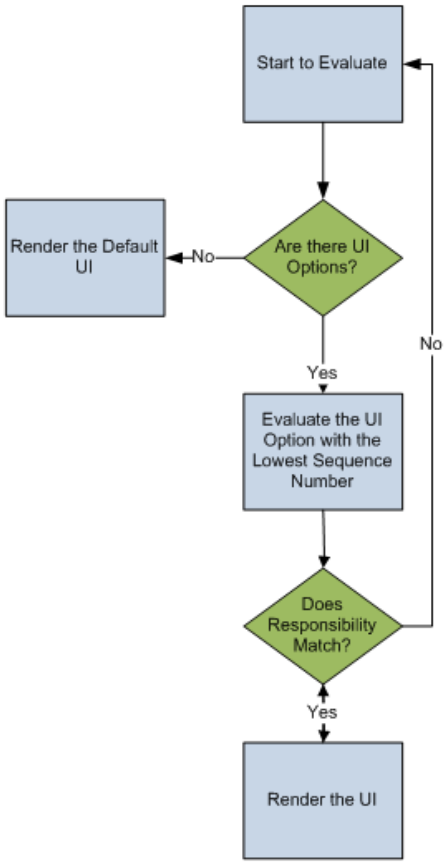

Figure 4. Evaluation Logic for Assigning a Siebel Configurator User Interface

NOTE: The default SWT file templates are documented as Applet User Properties found in Siebel Tools for the applet Cfg CX Runtime Instance Frame (JS HI).

About the Siebel Configurator User Interface View

Using the User Interface view, you can define the page or series of pages that display during a configuration session. These pages display when the user configures a customizable product as part of creating a quote in the Quotes screen. They also display when a user selects a customizable product in an eSales Web page. The same pages display when an administrator validates a customizable product.

The pages that display during the configuration process are called selection pages. The user makes selections from these pages to configure the customizable product.

You can design the following features of selection pages:

a **Basic page layout.** You set the basic look and feel of the customizable product's selection pages by choosing a base theme. Several base themes are provided. You can also build your own base themes.

b **Method of presenting products.** There are several ways to present your products. You can present them all on one page, you can set up several tabbed pages, or you can set up a wizard to guide the user through pages sequentially. You select a product theme in the Versions tab. You can also build your own product themes.

c **What items a page will contain.** You can choose which items to display on a page using a grouping mechanism provided in the User Interface view. You can add to a group the items in a relationship, the attributes of the customizable product, its links, or its resources. Depending on the product theme, each group displays on a separate page.

d **How items will be selected.** When you add an item to a group in the User Interface view, you can choose among several user interface control types for it such as combo box, check box, radio button, quantity box, and text box. These UI control types determine how the user goes about selecting an item.

The user interface you design is part of the customizable product's current work space. When you release a customizable product, the user interface is stored as part of the released version.

About Themes for the Siebel Configurator

When you create a work space for a customizable product, you can select a base theme and product theme. These control the basic look and feel of the pages a user sees when they configure the product.

About the Base Theme

A base theme defines the home page for the customizable product. It is the container within which the product theme pages display. The base theme has a thread bar you can use to navigate between product theme pages. Four types of base theme are provided:

- **Base Theme with Auto Pricing.** When the user selects an item, its price is updated immediately.

  This base theme has the following variations:

  - **Base Theme with Auto Pricing with Open UI.** When the user selects an item, the item price is updated immediately, this includes styling for Open UI.
  - **SIS OM Base Theme with Auto Pricing.** The theme provides additional fields within the container that are relevant to the communications industry.

- **Base Theme with Manual Pricing.** The pricing of items is not updated until the user clicks Update Price.

  This base theme has the following variation:

  - **SIS OM Base Theme with Manual Pricing.** The theme provides additional fields within the container that are relevant to the communications industry.

- **Base Theme with Multi Select JS.** The user can select many product components, and then click the Submit button to validate them all at once.
- **Base Theme with Quick Edit JS.** The user can turn validation on and off. Turning validation off saves time in configuring items.

Although both Multi Select and Quick Edit allow users to configure products more quickly, there are important differences between the two:

- **Constraint validation:**
  - A Multi Select user interface can invoke constraint validation for certain user actions, such as page refresh or selections using combo boxes. For example, when a user drills down on a child customizable product or changes tabs, the UI controls on the new page invoke object manager code which invokes the constraint engine and prompts the user to validate the Siebel Configurator session.
  - With a Quick Edit user interface, the user enables or disables the constraint engine by explicitly clicking a button.
- **Modification by the user:**
  - With a Multi Select user interface, the underlying logic for altering constraint engine invocation and constraint violation behavior can be changed by modifying the client side JavaScript files.

With a Quick Edit user interface, logic for altering the constraint engine cannot be modified by end users, because it resides in the object manager code.

Upgrade users: If you do not select a base or product theme, default themes are used. If you do not specify groups or controls, intelligent defaults are used. These defaults replace Configuration Assistant in release 6.x and lower.

About Product Themes

Product themes specify the style used to group items together on selection pages. You define which items appear on a page by defining groups in the User Interface view and adding products to the groups. Each group is displayed on a separate selection page.

Three basic product theme types are provided.

■ Tab Product Theme

The items in each group are displayed on a separate page, as shown in Figure 5. The Wireless Options group and the Features group each have a tab, and the user can move between pages by clicking the tab. A standard group of buttons, Save, Cancel, Done, and Finish It display in the header bar.

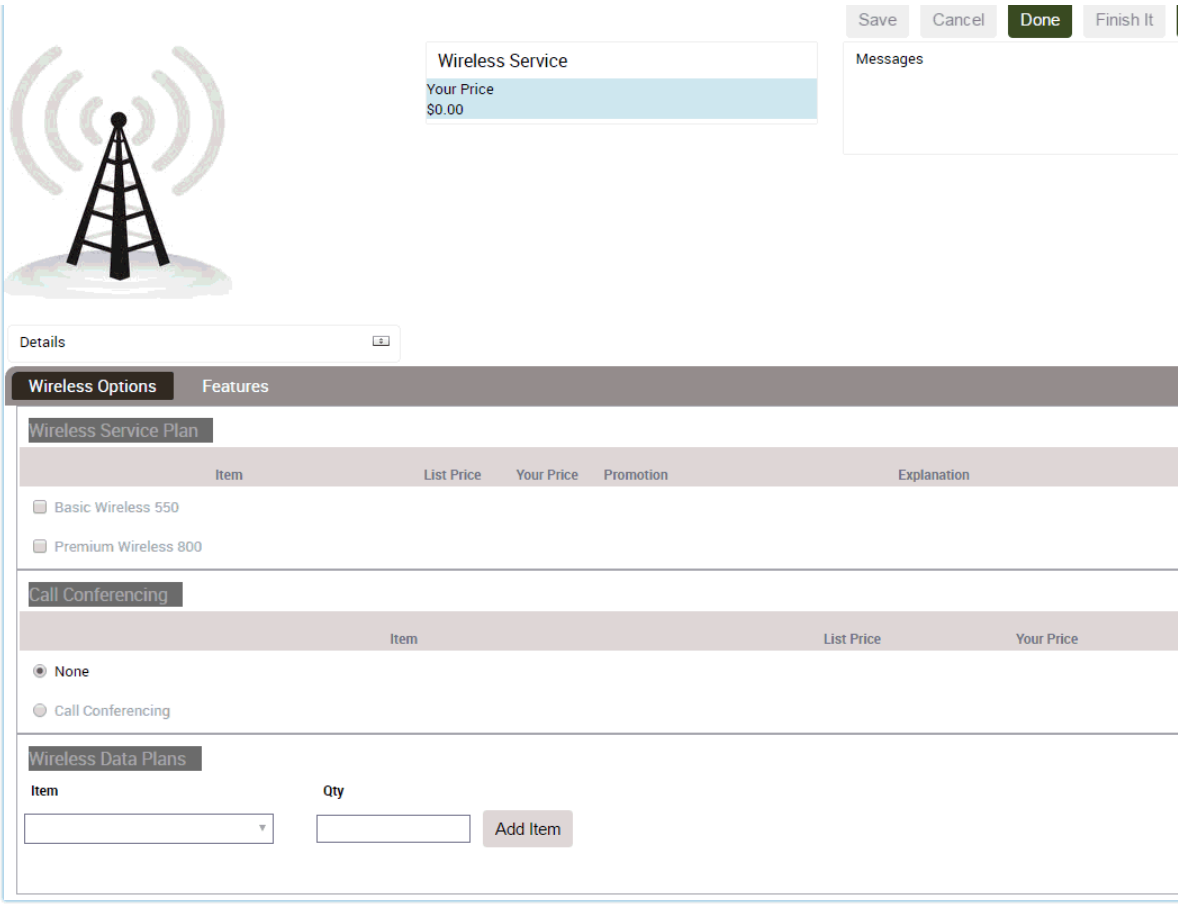

Figure 5. Tab Product Theme

■ Wizard Product Theme

The items in each group are presented on separate pages, as shown in Figure 6. The Wireless Options group is displayed by itself as the first page in the sequence. A standard group of buttons, Save, Cancel, Done, and Finish It display in the header bar. Previous and Next buttons, located within the page, allow the user to move between pages.

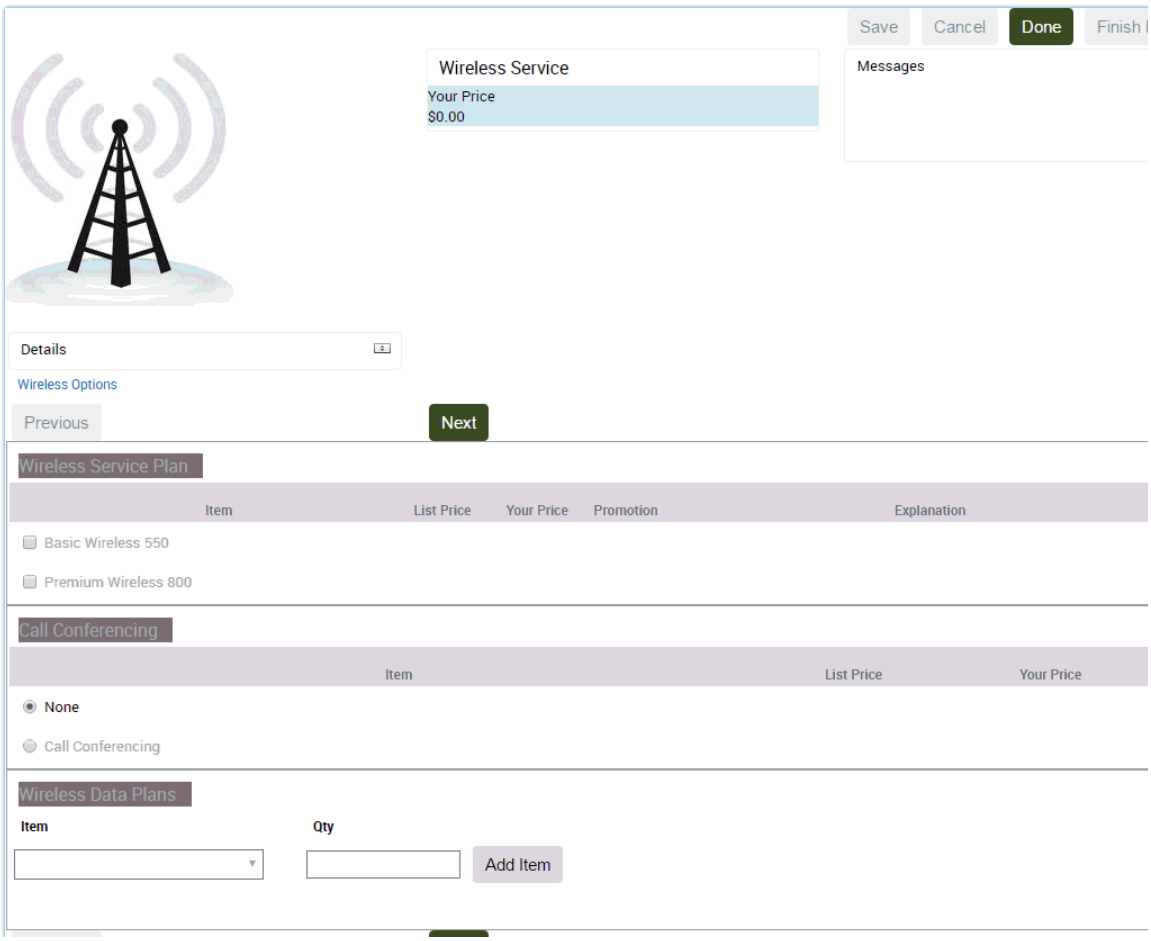

Figure 6. Wizard Product Theme

Single Page Product Theme

All the groups in the customizable product are presented on a single selection page, as shown in Figure 7. The tab pages used in the Tab theme and Wizard theme are stacked vertically one beneath the other to form the selection page. A standard group of buttons, Save, Cancel, Done, and Finish It display in the header bar.

NOTE: The Single Page Product theme does not support multilevel product structures.

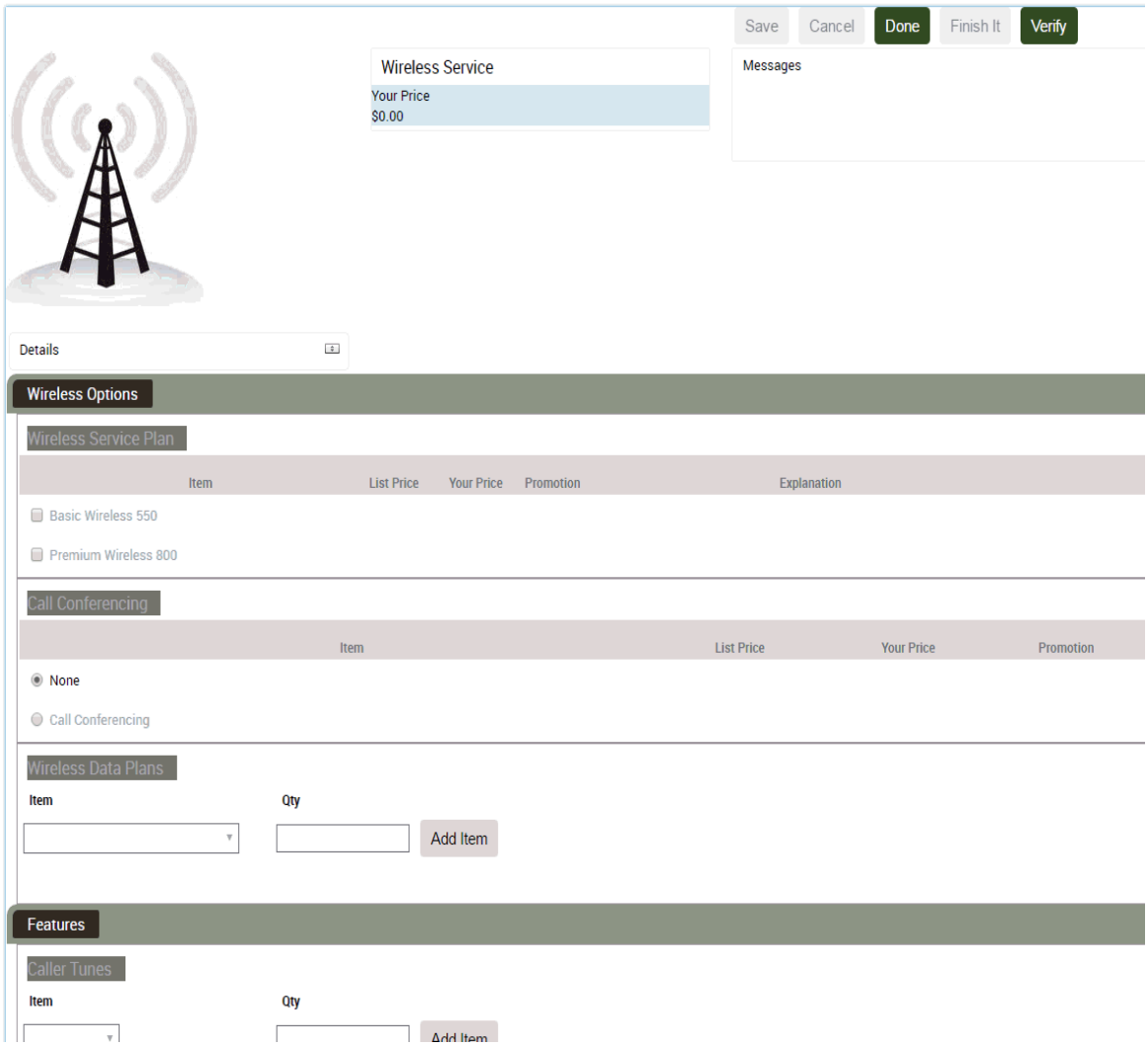

Figure 7. Single Page Product Theme

About Creating a Menu-Based Siebel Configurator UI

A special set of themes and UI controls is provided to build a menu-based interface:

- **Base Theme with Menu.** This theme must be selected as the base theme. This theme provides auto-repricing. No base theme is provided to select manual repricing.
- **Menu Product Theme.** This theme must be selected as the product theme. It displays relationships and their contents. It does not display attributes, resources, or linked items.
- **Standard Menu Group Theme.** This theme must be selected for all groups except a summary page group.
- **Summary Menu Group Theme.** Assign this theme to a group when you want to display a summary page. For more information on summary pages, see "Adding a Summary Page to the Siebel Configurator User Interface" on page 159.
- **Check Box For Menu Theme.** Select this UI control to display a check box with price.

The menu-based interface displays each group as a menu item. When the user clicks on the group name, a pane opens and displays the items that can be selected in that group. When the user makes a selection, the selection is displayed after the group name. Figure 8 shows a menu-based selection page. Two groups have been defined, Wireless Options and Features. The user's selection is shown after each group.

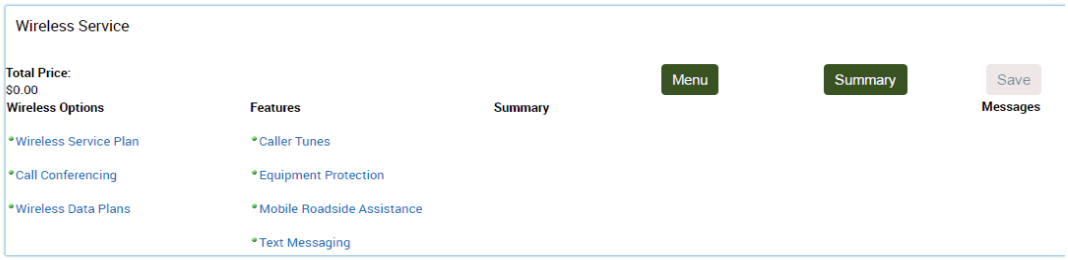

Figure 8. Example of Menu-Based Selection Page

Both group names are hyperlinks. If the user clicks on the Options group, a pane displaying the contents of that group opens as shown in Figure 9. The user can then select items from that group. When the user is finished, they click Menu to close the pane.

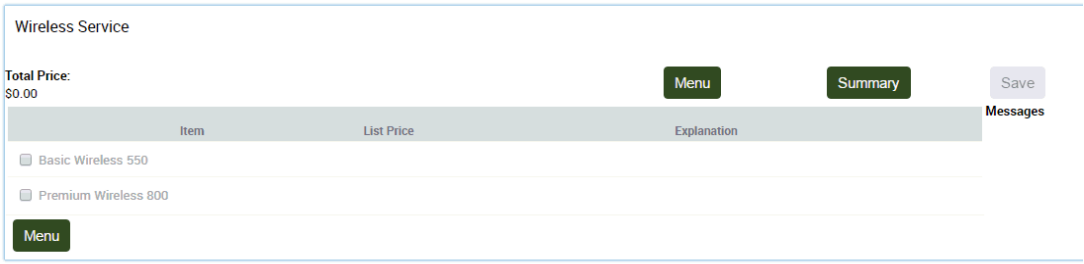

Figure 9. Option Package Group

Observe the following guidelines when using these themes and controls:

- The menu-based themes and controls can be used in conjunction with employee and partner applications. They cannot be used with customer applications, such as Siebel eSales.
- You cannot mix menu-based themes and controls with other types of themes and controls in a product. For example, if you select a menu-based base theme, you must also select a menu-based product theme as well as menu-based UI controls.
- A product with components that will be a component of another product with components cannot use menu-based themes and controls. For example CP2 is a product with components and a component of product CP1. CP2 cannot use menu-based themes and controls. Note that CP1 is not restricted from using menu-based themes and controls.
- Menu-based UI controls for links, resources, and attributes are not supported. Do not add links, resources, or attributes to groups.

About Siebel Configurator UI Groups

Groups are the way you define what items appear on a selection page. Depending on the product theme, each group you define causes a separate selection page to be created. For example, you want all the hard disks in a product with components to appear on the same page. You do this by defining a group in the User Interface view and then adding the Hard Disks relationship to this group. If you selected a tab type product theme, the hard disks display as a selectable page tab. When the user clicks on the tab, a selection page displays containing only the hard disks.

The User Interface view lets you create groups that contain relationships, attributes, resources, or links.

When you create a group, you can choose a group style. The group style defines the details of how a group will display.

About Siebel Configurator UI Controls

A user interface control determines how an object is displayed for selection. For example, a radio button control, displays a list of items with a button for each one. You choose a control as part of adding an item to a group in the User Interface view.

The User Interface view provides several types of UI controls:

- **Combo Box.** A combo box is a drop-down menu. Items are hidden from view until you click the down-arrow to open the menu and make selections. There are two types of combo box:
  - **Single selection.** The user can select only one value from the drop-down menu.
  - **Multiple selection.** The user can select multiple values from the drop-down menu using an Add Item button.
- **Check Box.** Items display as a list. Each item has a check box next to it. When you select an item, a check mark appears in the check box. You can select more than one item.
- **Radio Button.** Items display as a list. Each item has a button next to it. When you select an item, a dot appears in the button. You can select only one item. If you select another item, the previous item's button clears, and the current item displays a dot in the button.
- **Quantity Box.** A box displays next to the item in which the user enters or edits the quantity. The user then clicks elsewhere in the page to update the quantity.
- **Text Box.** A read-only box displays next to the item. The box contains the value of the displayed item. Use this UI control to display the current value of resources or linked items.
- **Edit Box.** A text box displays next to the item. The text box contains the value of the displayed item. You can edit the value. Use this control when you want users to be able to manually enter or edit attribute values.

Keep in mind the following factors when choosing a user interface control:

- For items added from a relationship, what are the cardinalities? If the minimum and maximum cardinality for the relationship is 1, this means only one item can be selected. The radio button or single-selection combo box can be used because it allows only one selection. If the relationship cardinality allows more than one selection, you must choose a UI control that allows multiple selections such as a check box or multiple-selection combo box.
- For attribute items, select a UI control that matches the attribute type. For LOV attributes use a combo box. For a range of values attribute or a single-value attribute, use a quantity box or text box.
- For resource items and for linked items, use a text box.

Table 13 on page 149 describes the specific UI controls available in the User Interface view. In the Multiple Items column, Yes means that the UI control allows selection of more than one item from a list. No means you can select only one item.

For all controls, excluded items display unavailable. For example, if you assign a radio button control to a relationship, excluded items display with the radio button grayed out so that it cannot be selected.

You can revise this so that excluded items do not display at all. You do this by assigning the predefined Excluded UI property to an item. You assign the Excluded UI property in the Properties view.

Table 13. User Interface view UI Controls

UI Control Type	Select Multiple Items?	Description
Combo Box	No	Selected item highlighted.
Combo Box with Add Button	Yes	Selected item highlighted. Unselected items have Add button.
Combo Box with Price	No	Selected item highlighted. Displays price.
Combo Box with Price and Quantity	No	Selected item highlighted. Displays price. Allows user to enter quantity.
Combo Box with Add Button and Price	Yes	Selected item highlighted. Unselected items have Add button. Displays price.
Combo Box with Update Quantity button	No	Selected item highlighted. User can specify a quantity for the selected item.
Check Box with Price	Yes	Displays item price.
Check Box without Price	Yes	Price not displayed.
Radio Buttons with Price	No	Displays item price.
Radio Buttons without Price	No	Price not displayed.
Quantity Box	User enters quantity	Update Quantity button displays next to item name.
Quantity Box with Price	User enters quantity	Displays item price. Update Quantity button displays next to item name.
Text Box with Price	No	Read-only. Displays item price.
Text Box	No	Read-only.
Combo Box for Attribute	No	Selected item highlighted.
Edit Box for Attribute	No	Displays value of attribute. Can be edited.
Radio Button for Attribute	No	User can select one attribute value.
Text Box for Attribute	No	Read-only. Displays value of attribute.
Linked Item	No	Read-only. Displays value of linked item.
Resource	No	Read-only. Displays value of resource.

About Pricing Integration with Siebel Configurator

Siebel Pricer works in conjunction with Siebel Configurator to provide updated pricing information during a configuration session. There are two methods for obtaining updated pricing information: automatic and manual. You select which method to use when you select a base theme.

Automatic price updates is the default method and is the method used by the default base theme. Base themes that provide manual price updates are labeled as such when displayed in the dialog box where you select the base theme. Unless labeled otherwise, base themes use the automatic price update method.

With automatic price updates, the pricing of the entire product with components is updated when the session starts, each time a new solution is created during the session, and when the session ends. When the user picks a product, the price of the product displays automatically.

With manual price updates, the pricing of the entire product with components is updated when the session starts, when the user clicks the Check Price button, and when the session ends. During the session, prices are not updated automatically when the user picks a product. Users must click the Check Price button to obtain the prices of the products they select.

When a price update occurs during a configuration session Pricer pricing elements trigger in the following order:

1. Component-based pricing adjustments
2. Attribute-based pricing adjustments
3. Pricing procedures. Only pricing that applies to the individual product triggers. Aggregate and volume discount pricing does not trigger.

When the session ends, Pricer pricing elements trigger in the same order.

When building a customizable product, use automatic price updates. Switch to manual price updates only if performance becomes too slow. The sequence of events after the user selects a product is as follows:

- Siebel Configurator engine computes a new solution
- Siebel Configurator engine forwards the solution to Pricer
- Pricer returns pricing for all items in the solution
- Siebel Configurator redisplays the selection page

If you do not need interactive pricing, consider switching to manual price updates.

Creating Custom UIs for Customizable Products

End users use Siebel Configurator to choose the features of products with attributes and products with components. You can customize the Siebel Configurator interface for both of these.

To create custom UIs for products with attributes

1. Navigate to the Administration - Product screen, then the Product Classes view.
2. In the Product Classes list, select and lock the desired product class.
3. In the Versions list, click the Work Space version.
4. Navigate to the User Interface view.
5. Design the user interface by following the instructions in this chapter.

   All products associated with this product class inherit this interface.

To create custom UIs for products with components

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired customizable product.
3. In the Versions list, click the Work Space version.
4. Navigate to the User Interface view.
5. Design the user interface by following the instructions in this chapter.

Process of Creating a Custom Siebel Configurator User Interface

To create a custom Siebel Configurator user interface, perform the following tasks:

- "Creating the Siebel Configurator User Interface Record" on page 151
- "Selecting the Base and Product Themes" on page 152
- "Enabling the Multiselect Feature for Siebel Configurator" on page 152
- "Enabling the Quick Edit Feature for Siebel Configurator" on page 154
- "Grouping Items onto Pages of the Siebel Configurator User Interface" on page 157
- "Adding a Summary Page to the Siebel Configurator User Interface" on page 159
- "Assigning Siebel Configurator Interfaces to Users" on page 160

CAUTION: In the Customizable Product UI layer, the type of the user interface controls must match with the type of the user interface option and the base theme type of the user interface option. If the types do not match, the user interface will be rendered incorrectly or there will be runtime errors. For example if you select a Customizable Product menu base theme, then you must only select relationship or attributes controls for menu theme.

Creating the Siebel Configurator User Interface Record

First, create the user interface record.

To create the user interface record

1 Select and lock the desired product, and navigate to the User Interface view, as described in "Creating Custom UIs for Customizable Products" on page 150.

2 In the User Interface list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Option Name	Enter a name for the user interface.
Option Type	Select High Interactivity. **NOTE:** This value must be used in Open UI.
Sequence	Enter a sequence number, which is used to decide which user interface to assign to users who are assigned multiple user interfaces. For more information, see "Assigning Siebel Configurator Interfaces to Users" on page 160.

Selecting the Base and Product Themes

User interface themes are templates that control the basic look and feel of the selection pages that users see when configuring a customizable product. Base themes control basic page design and product themes control the method used to present product choices. For more information, see "About Themes for the Siebel Configurator" on page 141.

This task is a step in "Process of Creating a Custom Siebel Configurator User Interface" on page 151.

To select the base and product themes

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Products list, select the desired product.

3 In the Versions list, click the Workspace hyperlink.

4 Click the User Interface view tab.

5 In the User Interface list:

   a In the Base Theme field, select the desired base theme.

   b In the Product Theme field, select the desired product theme.

Enabling the Multiselect Feature for Siebel Configurator

The multiselect user interface (UI) for Siebel Configurator allows the user to select many product components, and then click the Submit button to validate them all at once. You enable this feature by selecting the appropriate base and product theme.

If you do not enable this feature, the Siebel Configurator engine validates each product component individually when it is added. This means that the user has to wait for the UI to be refreshed and for the engine to validate the product after each component is added. Power users of Siebel Configurator, who already understand which components are valid, prefer to add many components and validate them all at once.

After the user clicks Submit, Siebel Configurator does the following:

- Fulfills all requests that are valid and displays the product with all these features incorporated.
- Lists all the conflicts for requests that are invalid.
- If any items submitted are invalid, the configuration reverts to the last successful submission.

Then the user can resolve conflicts by selecting Proceed for some features, and clicking the Resolve Conflicts button to submit these selected features for validation.

Alternatively, the user can not resolve these conflicts and instead make new selections in Siebel Configurator for these product features.

At any time during the configuration session, the user can click the Reset button to restore the configuration to what it was at the time of the last successful submit operation.

The multiselect UI base theme restricts the UI control themes that can be used. Use one of the following control themes:

- With the Web template SWT JS (RELATIONSHIP):
  - Check Box with or without price
  - Combo Box with price JS (eCfgControlComboPriceJS.swt) (no add button)
  - Quantity Box JS (eCfgControlQuantityJS.swt) quantity with or without price
  - Radio Buttons with or without price
- With the Web template SWT JS (ATTRIBUTES):
  - Combo Box for Attribute JS (eCfgAttributeComboBoxJS.swt)
  - Radio Button for Attribute JS (eCfgAttributeRadioJS.swt)
  - Edit Box for Attribute JS (eCfgCompAttributeEditJS.swt)

To enable the multiselect feature for Siebel Configurator UI

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the desired product.
3. In the Versions list, click the Workspace hyperlink.
4. Click the User Interface view tab.
5. In the User Interface list, in the Base Theme field, select eCfgBaseAutoRepriceMultiJS

Enabling the Quick Edit Feature for Siebel Configurator

This task is a step in "Process of Creating a Custom Siebel Configurator User Interface" on page 151.

The quick edit feature gives users flexibility to speed up configuration by turning off validation by the constraint engine, so the user does not have to wait for each change to be validated. Configuration is launched with validation turned off. In the Siebel Configurator user interface, users can:

- Turn quick edit off and turn validation on by clicking the Turn Off Quick-Edit button.
- Turn quick edit on and turn validation off by clicking the Turn On Quick-Edit button.

This feature allows more experienced users to make changes more quickly by using Siebel Configurator with validation turned off. Less experienced users can turn validation on to avoid making errors.

If Quick-Edit is turned on and the user clicks the Done button to exit Siebel Configurator, a message appears prompting the user to validate the configuration. The user can validate the configuration by clicking the Turn Off Quick-Edit button to enable the constraint engine.

NOTE: Turning quick edit on and off only applies to the constraint engine. User updates always go through eligibility and compatibility, promotion, and cardinality checks, regardless of whether quick edit is on or off.

Scenario for Using the Quick Edit Feature

For example, a business sells a complex frame relay product.

Some sales agents have extensive experience configuring this product, so they know which components are compatible, and they want the ability to add many components and validate them all at once. Other sales agents have little experience configuring this product, and they want the product to be validated after each component is added.

The product administrator creates the Siebel Configurator user interface using the Quick Edit base theme. After completing the design of the user interface, the administrator associates it with the responsibility of all these sales agents.

When sales agents launch Siebel Configurator, validation is turned off. Less experienced agents click the Turn Off Quick Edit button, so each component they add is validated as they configure the product.

More experienced agents add several components that are not validated individually. Then they click the Turn Off Quick Edit button to have the engine validate all the components that they have added up to that point. Then they click the Turn On Quick Edit button and add another group of components that are not validated individually.

Error Messages for Quick Edit

If a sales agent adds several components that are not validated individually and then clicks the Turn Off Quick Edit button, the Siebel Configurator might display error messages:

- If the components are compatible and the configuration is valid, no message is displayed.

- If a required component is missing and the configuration is incomplete, Siebel Configurator adds missing products to create a valid configuration and displays a message in the Messages box saying which products are added by the engine.
- If incompatible components are added, Siebel Configurator displays a conflict message and the user has two options:
  - Click Undo to go back to the previous page and correct the error.
  - Click Proceed. Siebel Configurator might display one incompatible product in red or might remove one incompatible product.

Modifying the Web Template for Quick Edit

To enable the quick edit feature, use the following procedure to modify the Web template.

NOTE: This topic does not apply if you are using Open UI.

To modify the Web template for quick edit

1. Navigate to the directory *siebel install directory*\webtempl.
2. Open the Web template file eCfgTopLevelButtonsJS.swt for the UI control using a text editor such as Notepad.
3. Edit the file by adding the lines highlighted in boldface below:

```
<table datatable="0" summary="" border="0" bgcolor="white" width="100%">
    <tr>
    <td nowrap>
        <swe:control id="swe:EnableEngine" CfgUIControl="EnableEngine" CfgHtmlType="MiniButton"
        InvokeMethod="EnableEngine"/>
    </td>
    <td nowrap>
        <swe:control id="swe:CheckPrice" CfgUIControl="Check the price" CfgHtmlType="MiniButton"
        InvokeMethod="CalculatePriceCX"/>
    </td>
    <td nowrap>
        <swe:control id="swe:Save" CfgUIControl="Save" CfgHtmlType="MiniButton"
        InvokeMethod="SyncInstance"/>
    </td>
```

Verifying the Signals for Quick Edit

To enable the quick edit feature, use this procedure to verify that the following signal definitions are present:

- PrepareEnableEngine
- ReEnableiLogForQuickEdit
- UpdateUIForQuickEdit

- PrepareDisableEngine

To verify the signals for quick edit

1. Navigate to Administration - Order Management screen, then the Signals view.
2. In the Signals list, confirm that the PrepareEnableEngine signal was created.
3. In the Versions list, click on the latest version of the signal to drill down to the Actions list.
4. In the Actions list, confirm that the fields of the Action record have the values in the following table.

Field	Value
Sequence	1
Service Type	Business Service
Service Name	Remote Complex Object Instance Service
Service Method	PrepareEnableEngine

5. Repeat Step 1 to Step 4 for the signals ReEnableiLogForQuickEdit, UpdateUIForQuickEdit, and PrepareDisableEngine.
6. Clear the cache.

Setting Up the Quick Edit Feature

To set up the quick edit feature, design the Siebel Configurator user interface in the usual way. Choose Base Theme with Quick Edit JS as the base theme for the interface. Associate this interface with the appropriate responsibility.

To set up the quick edit feature for the Siebel Configurator UI

1. Navigate to the Administration - Product screen, and then the Product Definitions view.
2. In the Products list, select the desired product and lock it if necessary.
3. In the Versions list, click the Workspace hyperlink.
4. Navigate to the User Interface view.
5. In the User Interface list, in the Base Theme field, select the template eCfgBaseAutoRepriceQuickEditJS.swt.

Disabling the Proceed Button

The Proceed button is available after the user clicks the Turn Off Quick Edit button to display conflicts among the components already added.

If there are incompatible components and the user clicks Proceed, Siebel Configurator might display one incompatible component in red or might remove one incompatible component. The product administrator can disable the Proceed button, if users might be confused by this behavior.

Disabling the Proceed button applies only when the Quick Edit mode is turned on. Because the user can make multiple selections before validating the configuration, the constraint engine cannot determine which user action caused the conflict, so the Proceed button is disabled. However, when the quick edit mode is turned off, each user action is evaluated by the constraint engine, so the engine knows which action causes a conflict; in this case, the Proceed button is enabled.

To disable the Proceed button

1. Navigate to the Administration - Product screen, and then the Product Definitions view.
2. In the Products list, select the root product and lock it if necessary.
3. In the Versions list, click the Workspace hyperlink.
4. Navigate to the Properties view.
5. Add a new record to the Properties and complete the necessary fields as described in the following table.

Field	Comments
Name	Enter *DisableProceed*.
Value	Enter *Y*.

When the sales agent configures the product, the Proceed button is disabled and grayed out.

Grouping Items onto Pages of the Siebel Configurator User Interface

Groups are the way you define which components or attributes of a product go on which selection pages. Depending on the product theme, all the items in a group display on one page. These pages display when the user selects the item for configuration.

For example, you design a customizable product that includes a computer monitor and a service plan. The user can select from among 4 monitors and 3 service plans. To display monitors and service plans on separate selection pages, you would create one group for monitors and one for service plans.

All the relationships, resource definitions, and linked items in a customizable product are listed in the User Interface view. Each item has an Add Item to Group button.

The attributes of the product are also listed. For products with components, these are not the attributes defined for items in relationships. These are the attributes that the whole product with components inherits from the class to which it belongs in the product table.

Setting up groups requires two steps:

a. Create a group for each selection page.
b. Add items to the groups.

This task is a step in "Process of Creating a Custom Siebel Configurator User Interface" on page 151.

To create a group

1 In the User Interface list, click the name of a user interface.

The User Interface Group list appears.

2 In the User Interface Group list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Group Name	Enter a name for the group. This name displays on the selection page. Use the product theme to guide how you choose group names. For example, if you use a tab theme, name the group according to what you want to appear on the page tabs.
Group Theme	Select the group theme, which controls the layout of the group within the base theme. You can select the Standard Group Theme. You can also select the Grid Layout Theme when you want to display controls across multiple rows and columns. For information about using the Grid Layout Theme, see "The Grid Layout Group Theme" on page 162.
Sequence	Enter a number specifying the order in which pages display. The group with sequence = 1 displays first in the page series.
Description	Enter a description of the group for your own use. The description does do not display on the selection page.

3 Repeat Step 2 until you have created all the desired groups.

Each group definition corresponds to one selection page. After you define a group, add the items you want to display on its corresponding page.

To add items to a group

1 Select the desired group.

2 In the explorer display of the product, select the desired item and click Add Item.

The item appears in the Group Item list, after the Group list.

3 Complete the remaining fields in the Group Item list, described in the following table.

Field	Comments
Name	Displays the name of the item you added. For Relationship items, this field contains the product display name. For Attribute, Resource and Linked Items, this field contains the item name. This field displays in the selection page.
Type	Displays whether the item is from a relationship, is a resource, a linked item, or is an attributes.
Sequence	Enter a sequence number to control the order of display of relationships within a group. The item with Sequence =1 is displayed first in the group.
UI Control	Select a UI control, such as radio button or check box for the item or for the items in a relationship. For more information, see "About Siebel Configurator UI Controls" on page 148.

4 Repeat Step 3 for each item you want to add to the group.

Adding a Summary Page to the Siebel Configurator User Interface

You can add a summary page that shows users how they have configured a product. This page displays the relationships from which the user has made selections along with attribute values. It also displays all the selected products the user has chosen with an editable quantity. If needed, you can make this quantity read-only by customizing the summary tab .swt.

Each relationship is a hyperlink. When the user clicks on the relationship name, the selection page containing that relationship displays, and users can revise their selections.

Figure 10 shows an example of a summary page. The first portion of the page lists the relationships and the attribute values that have been selected. The last portion of the page shows the items that the user has selected from each relationship.

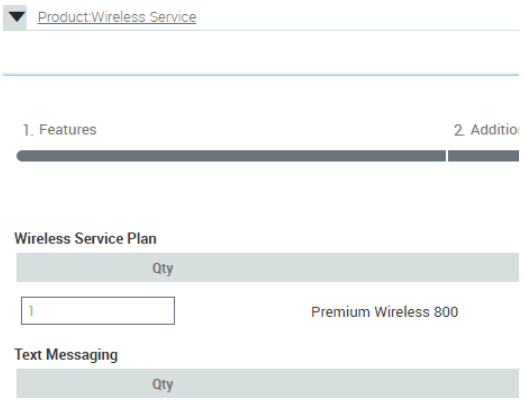

Figure 10. Example of a Summary Page

This task is a step in "Process of Creating a Custom Siebel Configurator User Interface" on page 151.

To add a summary page

1 In the User Interface view, create a group, as described in "Grouping Items onto Pages of the Siebel Configurator User Interface" on page 157, and complete the fields with the values described in the following table.

Field	Value
Group Name	Enter the name that will be the tab or page title name. For example, enter Summary.
Group Theme	Select Summary Group Theme.
Sequence	Enter a number to set the sequence of the summary page. To make the summary page the last page in the series, enter a sequence number that is higher than the number of the other groups

2 Do not add items to the group. The application generates the summary page automatically.

Assigning Siebel Configurator Interfaces to Users

After you have created custom user interfaces for all your users, you must assign each interface to the appropriate users.

This task is a step in "Process of Creating a Custom Siebel Configurator User Interface" on page 151.

You can create a number of Siebel Configurator user interfaces for the same customizable product and assign them to different users. This can be useful for the following reasons:

- You can create a simple user interface for customers buying the product through the Web and an interface that allows quicker data entry for call center agents.
- You can create a user interface with only a small number of options for salespeople doing indicative quoting, who need to select only a limited number of product options to determine price. You can create a user interface with all available options for technical designers, who need to select all product options before submitting the order.

Because several user interfaces use the same product model, maintenance is reduced.

You assign a user interface to users by associating it with a responsibility. Users have responsibilities that determine what views they can see. If you set up your application for the first example above, you would have the responsibilities Web Customers and Call Center Agents. For the second example above, you would have the responsibilities Salesperson and Technical Designer. For more information about responsibilities, see *Siebel Security Guide*.

If you leave the responsibility blank, the user interface is assigned to all users. If you add another option with a specific responsibility to override the blank responsibility for some users, you must make the sequence number for this specific responsibility lower than the sequence number for the blank responsibility.

You can also control the UI option using "SetUIOption Method" on page 460.

To assign a Siebel Configurator user interface to users

1. Navigate to the Administration - Product, then the Product Definitions screen.
2. In the Products list, select and lock the desired customizable product.
3. In the versions list, click the Work Space version.
4. Navigate to the User Interface view.
5. Add records to the User Interface list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Option Name	Enter the name of a custom user interface.
Option Type	Select high interactivity.
Sequence	Enter the sequence in which this record is checked. When a user clicks Customize, the application checks the records in this list in order of sequence, and it assign the user the first record it finds that is assigned to the user's responsibility and supports the user's browser.
Responsibility	Enter the responsibility that this user interface is assigned to.

The Grid Layout Group Theme

The following topics discuss the grid layout group theme:

About the Grid Layout Group Theme

You use the Grid Layout Group Theme to define a grid of rows and columns and specify where on the grid you display each control.

The logic of the grid layout is also included in the underlying Web templates, so developers can enhance it.

Figure 11 shows an example of a page created using the grid layout group theme.

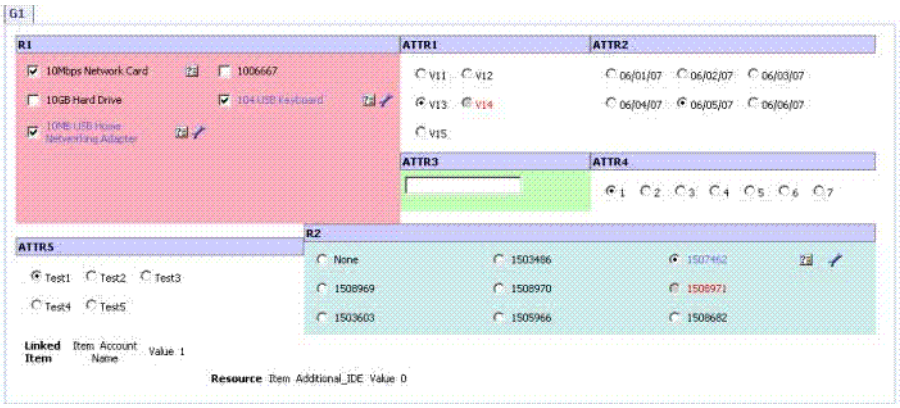

Figure 11. Grid layout Group Theme

Creating the Grid Layout Group User Interface

The grid layout group user interface allows you to control the location of each control in the group very precisely:

- You specify the number of columns used to display the controls.
- You specify where on the grid of rows and columns each control appears by entering the GridBegin and GridEnd property for that control.
- You do not specify the number of rows. The application determines the number of rows needed by the controls you define.

For example, you specify that the grid has three columns. To make a control appear on the second row, third column, you add that control as a Group Item and you give it the following properties and values:

- GridBegin = {2,3}
- GridEnd = {2,3}

Notice that the values are enclosed in curly brackets, the value of the row comes first, and the values of the row and column are separated by a comma. You must enter both a GridBegin and a GridEnd property for each control. In this example, the GridBegin and GridEnd properties have the same value, because the control occupies only one row-column combination on the grid.

You can also specify that controls occupy several rows and columns of the grid. For example, you have defined a grid with three columns. You want one of the controls to occupy the second and third columns of the fourth, fifth, and sixth rows. Give that control the following properties and values:

- GridBegin = {4,2}
- GridEnd = {6,3}

To create the grid layout group user interface

1. Create the group and add items to it, as described in "Grouping Items onto Pages of the Siebel Configurator User Interface" on page 157.

   When you create the group, in the Group Theme field, select GridLayout Group Theme JS.

   NOTE: If the GridLayout Group Theme JS is not available within the Group Theme pick applet, then manually create a record within the Group Theme pick applet and map it to the eCfgGroupGridLayoutJS.swt file.

2. Define the number of columns for the group:
   a. Select the group in the panel to the side.
   b. Click the Properties tab.
   c. Add a new record to the properties list, and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter *NumofCol*
Value	Enter the number of columns that you want. If you do not enter a number, the default value of 4 is used.

3. Define the locations of controls:
   a. Select each control in the panel to the side.
   b. Click the Properties tab.

c Add a new record to the properties list, and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter *GridBegin.*
Value	Enter the number of the row and column where you want this control to begin.

d Add another new record to the properties list, and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter *GridEnd.*
Value	Enter the number of the row and column where you want this control to end.

e Optionally, to customize the style of the cell, add another new record to the properties list, and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter *CellClass.*
Value	Enter the name of a style class if you want to customize the style of the cell, for example by specifying the font or background color. For example: ■ Add a new entry *.eCfgRadioGrid3 {background-color:#FAAFBE}* in the main.css file. ■ Enter *Cell Class* as the property and *eCfgRadioGrid3* as the value. ■ The cell will display with a pink background color. It is recommended that you not use the Cell Class property to control height or width attributes.

Guidelines for Using the Grid Layout Group Theme

Observe the following guidelines when using the Grid Layout Group Theme:

- The Grid layout feature is not supported for multi-level grandchild control templates, which recursively includes the group theme to show a grid table inside of another grid table.

- The grid layout theme should be used judiciously. The performance of the Siebel Configurator UI degrades with the number of controls displayed within the page.
- The grid layout theme can be used to control the display of all types of controls in the interface, including controls based on fields, relationships, attributes, linked items and resources.
- The following UI Control templates can be used with the Grid layout Group Theme.

  With the Web template SWT JS (RELATIONSHIP):

  - Check Box GridLayout JS (eCfgControlCheckGridLayoutJS.swt)
  - Radio Buttons GridLayout JS (CfgControlRadioGridLayoutJS.swt)
  - Combo Box with Price Attribute Grid JS (eCfgControlComboAddPriceAttribGridJS.swt)

  With the Web template SWT JS (ATTRIBUTES):

  - Radio Button GridLayout for Attribute JS (eCfgAttributeRadioGridLayoutJS.)

  NOTE: If these templates are not available within the UI Control pick applet, then manually create a record in the pick applet and map it to the appropriate swt files.

Example of Using the Grid Layout Group Theme

This topic gives one example of using the grid layout group theme. You might use this feature differently, depending on your business model.

The Product Administer has created a product called Car that has four attributes: Manufacturer, Model, Color, and Year. The administrator wants to display the attributes Manufacturer and Model grouped together in the first row, and the attributes Color and Year grouped together in the second row.

To use the Grid Layout Group Theme

1. Select the desired group.
2. In the Group Theme pick applet, make sure that the Group Theme is GridLayout Group Theme JS.

   NOTE: If the GridLayout Group Theme JS is not available within the Group Theme pick applet, then manually create a record within the Group Theme pick applet and map it to the eCfgGroupGridLayoutJS.swt file.
3. In the Group Item List applet, create the attribute controls.
   a. Create the attribute control for Manufacturer by completing the necessary fields. Some fields are described in the following table.

Field	Value
Name	Enter Manufacturer.
Type	Select Attribute.

Field	Value
Sequence	Enter 1.
UI Control	Select Edit Box for Attribute JS.

b Create the attribute control for Model by completing the necessary fields. Some fields are described in the following table.

Field	Value
Name	Enter Model.
Type	Select Attribute.
Sequence	Enter 2.
UI Control	Select Edit Box for Attribute JS.

c Create the attribute control for Color by completing the necessary fields. Some fields are described in the following table.

Field	Value
Name	Enter Color.
Type	Select Attribute.
Sequence	Enter 3.
UI Control	Select Edit Box for Attribute JS.

d Create the attribute control for Year by completing the necessary fields. Some fields are described in the following table.

Field	Value
Name	Enter Year.
Type	Select Attribute.
Sequence	Enter 4.
UI Control	Select Edit Box for Attribute JS.

4 Select the Properties Detail tab.

5 In the explorer display of the product, expand UI Options.

6 Select the desired Group and create a new record in the Properties list by completing the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter NumofCol.
Value	Enter 2.

7 In the explorer display of the product, expand the desired group.

8 Select Control Item.

9 Click the Properties tab.

a Select the Manufacturer Control Item and create a new record in the Properties list by completing the necessary fields. Some fields are described in the following table.

Field	Comments
GridBegin	Enter {1,1}.
GridEnd	Enter {1,2}.

b Select the Model Control Item and create a new record in the Properties list by completing the necessary fields. Some fields are described in the following table.

Field	Comments
GridBegin	Enter {1,3}.
GridEnd	Enter {1,4}.

c Select the Color Control Item and create a new record in the Properties list by completing the necessary fields.Some fields are described in the following table.

Field	Comments
GridBegin	Enter {2,1}.
GridEnd	Enter {2,2}.

d Select the Year Control Item and create a new record in the Properties list by completing the necessary fields. Some fields are described in the following table.

Field	Comments
GridBegin	Enter {2,3}.
GridEnd	Enter {2,4}.

When the product administrator validates the model, attributes Manufacturer and Model will be displayed in the first row and attributes Color and Year will displayed in the second row. Note that each control is processed and rendered to the grid table sequentially; from each side of the grid table, from start to end. The sequence is decided by Group Item sequence.

Tasks for Setting Up the Siebel Configurator Open UI User Interface

The Open UI user interface has several features that require configuration. The following topics describe how to set them up:

- "Setting Up the Grandchild Display of the Open UI Siebel Configurator User Interface" on page 168
- "Using the Attribute Inline Display Control in the Open UI Siebel Configurator User Interface" on page 171
- "Validating the User Interface for Customizable Products" on page 173

Setting Up the Grandchild Display of the Open UI Siebel Configurator User Interface

The grandchild display of the Open UI Siebel Configurator user interface allows you to display an arbitrarily deep model and instance hierarchy on a single page, with minimum refresh.

This feature must be used carefully, because performance degrades with the depth of the hierarchy displayed.

To turn on the grandchild display

1. Set the following UI Properties:
   - MultiChildrenPort = <the name of the port that has grand child> (at the parent CP group level)
   - Default Group = <Group Name> (At the child level)
2. Change the template, as follows:

```
<!-- Template Start: eCfgControlCheckPriceMultiLevelJS.swt -->


<swe:include file="eCfgPort_HeaderJS.swt"/>

<tr>

   <td colspan=3>

      <swe:for-each id="500" CfgLoopType="DomainAndChildren" startValue="1500"
count="Dynamic" iteratorName="101Id"
```

```
       Usage="CheckBox"

      CfgFieldList="CfgFieldName:Name, CfgUIControl:lblName, HtmlAttrib_width:310,
Default:Y*

                     CfgFieldName:RequireMoreChild, Default:Y*

                     CfgFieldName:List Price, CfgUIControl:lblListPrice,
DataType:DTYPE_CURRENCY, NeedRefresh:N, HtmlAttrib_align:center,
HtmlAttrib_width:80*

                     CfgFieldName:Current Price, CfgUIControl:lblYourPrice,
DataType:DTYPE_CURRENCY, HtmlAttrib_align:center, HtmlAttrib_width:80*

                     CfgFieldName:Explanation, CfgUIControl:lblExplanation,
HtmlAttrib_align:center*

                     CfgFieldName:Customize, CfgUIControl:lblCustomize,
HtmlAttrib_align:center"

                     >

          <swe:control id="swe:111Id + 4000" CfgHtmlType="CfgCheckBox"
ForceRefresh="Y"

         CfgJSShow="showCheckBox" CfgJSUpdateExclusion="updateExcludedItemForPort"
CfgJSUpdateSelection="updatePortItemsForCheckBox"/>



      </swe:for-each>


   </td>

</tr>


<!-- LM: for each selected product of the current relationship, include full UI
starting from group level -->

<tr>

   <td width="100%" colspan="3">

         <table width="100%" cellpadding="0" cellspacing="0" border="0">

           <swe:for-each id="500" CfgLoopType="Children" startValue="1500"
count="Dynamic" iteratorName="101Id">



                  <swe:switch>

                     <swe:case condition="Default, TestFieldValue, Operator:==,
FieldName:CanDrillDown, FieldValue:Y">
```

```
            <swe:for-each id="110" CfgLoopType="CurrentGroup" startValue="8100"
count="Dynamic" iteratorName="100Id">

               <tr>

                  <td width="10%">

                  </td>

                  <td width="80%">

                 <swe:include id="swe:151Id" CfgHtmlType="CurrentGroup"/>

                  </td>

                  <td width="10%">

                  </td>

               </tr>

        </swe:for-each>


                        </swe:case>

                        <swe:default>


   <swe:for-each id="110" CfgLoopType="Attribute" startValue="1100" count="Dynamic"
iteratorName="IncId" >

      <tr class="listRowOff">

         <td class="row" width="100%" class="AppletBlank">

            <swe:include id="swe:151Id" CfgHtmlType="Children"/>

         </td>

      </tr>

   </swe:for-each>


   <tr><td width="100%" class="AppletBlank"></td></tr>

   <tr><td width="100%" class="AppletBlank"></td></tr>


   <swe:for-each id="100" CfgLoopType="Port" startValue="2100" count="Dynamic"
iteratorName="IncId" >
```

```
        <tr class="listRowOff">
           <td width="100%" class="row" class="AppletBlank" >
              <swe:include id="swe:151Id" CfgHtmlType="Children"/>
           </td>
        </tr>
    </swe:for-each>

    <tr><td width="100%" class="AppletBlank"></td></tr>
    <tr><td width="100%" class="AppletBlank"></td></tr>

                    </swe:default>
                 </swe:switch>
              </swe:for-each>
           </table>
    </td>
</tr>
<swe:include file="eCfgPort_FooterJS.swt"/>
<!-- Template End: eCfgControlCheckPriceMultiLevelJS.swt -->
```

Using the Attribute Inline Display Control in the Open UI Siebel Configurator User Interface

The Attribute Inline Display control allows you to display attributes of child products on the same Siebel Configurator selection page as the parent product.

The control is a form with attributes of the child product. You can click tabs at the beginning of the control to display forms for different child products.

For example, a parent product might be Mobile Package, a package of options for purchasing cell phone service. Child products might be Mobile Subscription Options, Price Plans, and Accessories. When the user displays the selection page for Mobile Package, the Attribute Inline Display Control has a series of tabs that the user can click to display the attributes for these child products.

To set up the Attribute Inline Display control

1 For the Parent Product from which users can choose a Child Product:

a. Navigate to the Administration - Product screen, then the Product Definitions view and select the Customizable Product (for example, Mobile Package).

b. In the Versions list, click the Work Space version.

c. Click the User Interface view tab.

d. Select the UI Group where the child product is referenced (for example, Subscription Details).

e. `Select eCfgControlComboAddPriceAttribGridJS.swt` as the Group item theme.

f. For the Group name defined earlier (for example, Subscription Details), add new User Property named MultiChildrenPort, and provide the relationship name as its value (for example, Mobile Subscription Options).

2. For the child product whose attributes will be shown and selected:

a. Navigate to the Administration - Product screen, then the Product Definitions view and select the Customizable Product (for example, Mobile Subscription).

b. Navigate to User Interface view.

3. Create a new UI Group, and complete the fields with the values described in the following table.

Field	Value
Group Name	Grandchild Attr
Group Theme	Standard Group Theme JS
Sequence	10

a. Add all the attributes that you want to show as Group Items in this new UI Group, for example, Activate Immediately, Activated.

b. Navigate to User Interface view.

4. Create a new root product UI Property and complete the fields with the values described in the following table.

Field	Value
Default Group	Grandchild Attr

a For an attributes that you specified earlier, such as Activate Immediately, create a new UI Property and complete the fields with the values described in the following table.

Field	Value
Grandchild Enabled	Y

Create another new UI Property and complete the fields with the appropriate values, such as the values described in the following table.

Field	Value
Grandchild Type	combo

NOTE: The type is case sensitive and must be in all lower case. Other supported types are *text* and *radio*.

b For other attributes that you specified earlier, such as Activated, create a new UI Property and complete the fields with the values described in the following table.

Field	Value
Grandchild Enabled	Y

Create another new UI Property and complete the fields with the appropriate values, such as the values described in the following table.

Field	Value
Grandchild Type	Combo

c Release a New Version of the product

5 Release the Parent Customizable Product.

Validating the User Interface for Customizable Products

When you validate the user interface for customizable products, you can pick an option for validation in the workspace list applet. When you click Validate, Siebel Configurator tries to load the specified user interface regardless of the current user or browser. This allows you to test all the user interfaces for every different type of user.

For more information about validating products, see "Testing a Product with Components in Validation Mode" on page 331.

About Managing Item Groups

You can manage item groups in the following ways:

- "Editing Item Groups"
- "Deleting Item Groups"

Editing Item Groups

You edit item groups by selecting a group and then editing the group record or by editing items in the group's Group Item List.

In the group record, you can change the group theme or the order in which the group displays.

In the Group Item List for a group, you can edit records by changing their sequence of display or by changing the type of UI control used to display the item.

If you change the UI control for a relationship, make sure that the new control allows the user to exercise the full range of the cardinalities you have defined for the relationship.

Deleting Item Groups

Item groups are the mechanism you use to group items onto selection pages. Deleting a group removes the page and all its items from the collection of pages displayed in Siebel Configurator.

12 Siebel Configurator UI Properties

This chapter explains how to use the Properties view to modify the display of selection pages for customizable products. You must define a work space and have used the User Interface view to define selection-page layout before using the Properties view. It includes the following topics:

About Siebel Configurator UI Properties

A UI property is a named variable and its value. UI properties modify the display of an item in Siebel Configurator. You define a UI property by selecting the desired item and then defining one or more UI properties for it.

There are two types of UI properties:

- Predefined UI properties, such as Hide, are Siebel-provided UI properties that perform special functions. For more information, see "About Predefined UI Properties" on page 177.
- User-defined UI properties are those that you define and then insert into a Web template to control the display characteristics of an item. For more information, see "Using User-Defined UI Properties" on page 181.

The Properties view displays all the items in the customizable product:

- Product Name
- Attribute
- Relationship
- Group
- Linked Item
- Resource

To use the Properties view, you select an item in the customizable product and then define a UI property for the item in Item Display Properties. Each record in Display Properties has two fields. The first contains the name of the UI Property variable. The second field contains the value you want to assign to the variable. This value is what displays in the configuration session.

Observe the following guidelines when defining UI properties:

- The relationship between an item and a UI variable name is one-to-one. You cannot define multiple UI properties for an item, each with the same variable name.
- An item can have multiple UI property definitions, each for a different UI variable. An example of this would be an item that appears in multiple locations within a customizable product.

About Predefined UI Properties

Table 14 shows the predefined UI properties that you can use. These UI properties provide commonly desired ways to modify the display of items. You do not need to insert a variable name for these properties into a customizable product Web template. You only need to assign them to the desired item in the Properties view.

NOTE: For external images, the image must be stored in the Siebel installation directory in \Public\enu\Images\<filename>, where <filename> is the name of the file.

Table 14. Predefined UI Properties

Property Name	Value	Description
Excluded	Y	Cannot be defined on products within a relationship or attribute values. When defined on a relationship, prevents all excluded items in the relationship from displaying. If an item in a relationship is a product with components, does not prevent display of excluded products within that product with components. When defined on the product root, prevents excluded items from displaying throughout the product. When defined on the attribute definition, prevents excluded attribute values from displaying. This property is applicable for enumerated attribute definitions.
Hide	Y or N	When set to Y for any relationship, product or attribute, this value causes that item to be omitted from the UI selection pages during customization. This UI property is very useful with class-products. For child products in the class, you can hide portions of the product structure inherited from the class-product. This allows you to define the class-product structure and then tailor its display for each of the child products that inherit the structure. Can be defined on any part of a customizable product structure, shown on one side of the tree view that displays in the UI Properties administration view. Acts at the UI level, so if you add this UI property to hide the values that are selected in the existing quotes it does not remove these values from the structure, but only changes the display. This property hides the item statically, so it remains hidden. You hide products dynamically using a different property, described in “Creating Dynamic Siebel Configurator User Interface Controls” on page 182
Description	Enter a text string.	Define on relationships only. Enter the text exactly as it will display to the user.

Table 14. Predefined UI Properties

Property Name	Value	Description
Image	Images/ <filename>	Define on relationships only. The image displays beside of relationship header. The image is displayed full size.
LearnMore	Enter the full URL to the desired location	Use with relationships only. Do not use with component products, resources, attributes, or links. The words “Learn More” are displayed adjacent to the item and are a hyperlink to the URL you enter.
ProductHeader Image	Images/ <filename>	Define on product root only. Displays an image of the root product on every selection page. Image displays beneath item header, to one side of item labels. The default image area is 120x120 pixels square. Can only be defined on product root.
FullComputation	Y	When set to Y (Yes) and user makes an attribute selection, the Siebel Configurator engine updates the selection state of all the attribute values so that only selectable values are displayed. For example, if one of the values is excluded, it displays unavailable. Can cause performance reduction. This is the default. When set to N (No), the Siebel Configurator engine does not update the selection state of the other attribute values and displays all the values as selectable. For example, if one of the values is excluded, it does not display unavailable. If users select an excluded value, they receive a conflict message. Use this UI property when display of the selection state of attribute values is required. Define on attributes with LOV domains only.
Resource	Y	Define on attributes only. When this property is set, the attribute value will be treated as engine picked, which works similar to a resource. This property is commonly used when an attribute is used as a counter that will not be changed directly by a user and that will be recalculated every time regardless what previous value or state is.
Default Group	The name of a valid group	This property specifies the default group loaded in the grandchild display.

Table 14. Predefined UI Properties

Property Name	Value	Description
Grandchild Enabled	Y	Define this property at the appropriate attribute group item level in order to show attributes in line with products in the grandchild display.
Grandchild Type	combo, text, radio	Define this property at the appropriate attribute group item level in order to show attribute in line with products in the grandchild display. This property specifies the type of attribute user-interface control to load.
MultiChildrenPort[N]		This property is used by the CP UI service. It lets the UI service know that before presenting a given group or the CP itself, the UI service needs to fetch more details about the user property value and relationship. You can define this at the root level or at each group level for multiple relationships (Name: MultiChildrenPort, Value: Intrastate, Name: MultiChildrenPort1, Value: USA, Name: MultiChildrenPort2, Value: Canada, Name: MultiChildrenPort3, Value: Extended and so on, where Intrastate, USA, Canada, Extended are relationship names within the CP. You would need to define this for each CP UI definitions. If you customize the OOTB template and add .swt template code to access child component information (attribute or relationship products) then you must define the root item as a multi children port. This is needed because the UICache used, may not contain the complete information needed for the UI (obsolete or not cached UI information such as accessing all attribute values of all products in the group item). The disadvantage of using this is that there will be performance penalty if the total relationship domains affected by these user properties are large. If you use this user property several times, remember to give them names such as MultiChildrenPort, MultiChildrenPort1, MultiChildrenPort2 and so on. (You cannot use MultiChildrenPort0.)

Table 14. Predefined UI Properties

Property Name	Value	Description
.MatchNewItemsInSolution	Y	When you have an asset with child products, and when you modify the asset and change its configuration, the engine removes all existing child items and then adds all new child items that fulfill the constraint rule. The Action Code field value changes from Delete, to Add. When this user property is set to Y on the root customizable product, the engine maintains and updates the existing child items while fulfilling the constraint rule. **NOTE:** If you require the Action Code field of the original instance of the component to have the value Update, then the Cfg State Code in the component in the asset hierarchy requires the value Engine Picked.
Grandchild Sequence	An integer greater than 0	This property is used to sort the grandchild attributes for display. Use it with the Grandchild Enabled property. Configure the UI properties for all attributes with the names Grandchild Sequence and with the values as integers representing the sequence numbers. In the user interface, the attributes are shown in ascending order of the sequence set, using this UI property.

NOTE: For external images, the image must be stored in the Siebel installation directory in \Public\enu\Images\<filename>, where <filename> is the name of the file.

Using User-Defined UI Properties

After you give a name to a user-defined UI property, you can use the property by using the appropriate JavaScript syntax.

For example:

- The UI property name is *ExcludeAtt*.
- You can use the UI property name with the following JavaScript syntax:

  ```
  attPropSet.GetProperty(".ExcludeAtt")
  ```

  NOTE: You must include a period (.) before the name.

For an example of how user-defined UI properties can be used, see the use of the UI variable in the topic "Creating a New Base Theme Template" on page 197.

Defining a UI Property

A UI property is a named variable and its value. UI properties modify the display of an item in a customizable product. You define a UI property by selecting the desired item in a customizable product and then defining one or more UI properties for it.

If you are using an attribute from a line item integration component field to define the UI property, see "Binding an Attribute Value to a Line Item Integration Component Field" on page 74.

To define a UI property for a product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Navigate to the Properties view.
5. In the Explorer applet, click the name of the item for which you want to define a UI property.
6. In Properties view, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter the name of the UI property variable. The variable can be predefined or user-defined.
Value	Enter a value for the variable. The value can be predefined or user-defined.

Creating Dynamic Siebel Configurator User Interface Controls

User properties allow you to show or hide user interface controls, to make user interface controls read-only, or to show or hide images, based on the choices that the user makes in Siebel Configurator.

For example, a business sells tee-shirts that can be personalized with the name and business name of the purchaser or that can be bought without personalization. The root product is a tee-shirt with attributes including the following: Personalize and Name. If the user selects the Personalize checkbox (which makes the Personalize attribute equal Y), then an additional Siebel Configurator tab is displayed that allows the user to enter the customer's name and the business name.

Dynamic Siebel Configurator user interface controls are discussed in the following topics:

- "About Using CfgEval()" on page 183
- "Dynamically Showing and Hiding Siebel Configurator Controls" on page 184
- "Example of Dynamically Showing and Hiding Siebel Configurator Controls" on page 186

- "Dynamically Making Siebel Configurator Controls Read-Only" on page 187
- "Dynamically Displaying Images in Siebel Configurator" on page 187

About Using CfgEval()

Use CfgEval() to access instance information inside Siebel Configurator, for example, to whether a user has set an attribute or selected a component product.

CfgEval() returns a Boolean value and is currently supported within:

- The PickSpec UI property
- The Dynamic Show and Hide and Dynamic Read-Only.

Syntax Enhancements for CfgEval()

CfgEval() uses the standard syntax for product paths in Siebel Configurator. For more information, see Chapter 19, "Siebel Configurator Scripts."

CfgEval() also uses the following enhancements to the syntax for product paths:

- Use .xa[Attribute Name] to specify the attribute name
- Use .xf[Value] to return the attribute value
- Use .xf[Quantity] to return the product quantity

Here are examples of these enhancements:

- To return an attribute value:
  ```
  CfgEval($.[Root Product]#1.[Relationship]#[Component Product].xa[Attribute].xf[Value])
  ```
- To return the quantity of a component product:
  ```
  CfgEval($.[Root Product]#1.[Relationship]#[Component Product].xf[Quantity])
  ```

Using CfgEval() with Dynamic Show and Hide, Dynamic Read-Only, and Dynamic Display of Images

To use dynamic show and hide, dynamic read-only, or dynamic display of images, you must create a UI property on the root product whose value is an expression that uses the CfgEval() function and that evaluates to Y or N. For more information, see "Dynamically Showing and Hiding Siebel Configurator Controls" on page 184.

You can define the expression using the CfgEval() function in either of two ways:

- **Session Values.** Use the CfgEval() function to create simple expressions that reference session values such as the product quantity and attributes that the user selects. Here are some examples of how to use this function:
  - Check whether the user has selected an attribute on the Root Product:
    ```
    CfgEval($.[RootProduct]#1.xa[AttributeName].xf[Value])='ABC'
    ```

- Check whether the user has selected an attribute on the Child Product:
  `CfgEval($.[RootProduct]#1.[RelationshipName]#[ChildProductName].xa[AttributeName].xf[Value])='ABC')`
- Check the quantity of a child product:
  `CfgEval($.[RootProduct]#1.[RelationshipName]#[ChildProductName].xf[Quantity]) = ‘2'`
- To return an attribute value of a child product:
  `CfgEval($.[RootProduct]#1.[RelationshipName]#[ChildProductName].xa[AttributeName].xf[Value])`
- To return the quantity of a child product
  `CfgEval($.[RootProduct]#1.[RelationshipName]#[ChildProductName].xf[Quantity])`

For information about the syntax used in these functions, see “About Product Path in Siebel Configurator Scripts” on page 305.

- **Procedural Conditional Variables.** Use the CfgEval() function to reference procedural conditional variables to create more complex expressions, which are useful when you want to dynamically show or hide controls based on linked items, resources or compound expressions. For more information about procedural conditional variables, see “Procedural Condition Templates” on page 259.

  Here is an example of the format needed to use the CfgEval() function to reference a procedural condition: `CfgEval(Procedural Condition Constraint Name)`.

  Procedural conditional variables do not work with the Quick Edit or Multi-Select UI, because the constraint engine is disabled.

If the CfgEval() references the quantity of a child product and if there is more than one instance of this child product in the product, then CfgEval() computes the sum of the quantity in all the instances of the child product.

User actions that change the value of the CfgEval() expressions cause a page refresh, if controls must be hidden, shown, or inactivated.

Dynamically Showing and Hiding Siebel Configurator Controls

To dynamically show and hide Siebel Configurator controls, perform the following high-level steps:

1. On the root product create a UI property with the name CfgUIStateModel and with the value Y.
2. On the root product, create a UI property with some given name, such as *MyExpression*, and with a value that is an expression that evaluates as Y or N. Define this expression using the CfgEval() function. For more information, see “About Using CfgEval()” on page 183.

3 On UI controls for this product (such as relationships, groups, products, tabs, attributes, linked items or resources) create UI properties with the name DynamicHide plus the name you gave to the expression. For example, the name might be *DynamicHide MyExpression*. These controls are hidden when the expression evaluates to Y and shown when the expression evaluates to N.

NOTE: Dynamic show and hide is not supported for inline attributes.

To dynamically show and hide Siebel Configurator controls

1 On the root product, create a UI property with the name CfgUIStateModel and with the value Y:

a Navigate to the Administration - Product screen, Product Definitions view.

b Lock the Product you want to change.

c Drill down on the Work Space version of the Product you want to change.

d Navigate to the Properties view.

e In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *CfgUIStateModel*.
Value	Enter *Y*.

2 On the root product, create a UI property with some given name, such as *MyExpression*, and with a value that is an expression that evaluates as Y or N:

- In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter a name for the expression, such as *MyExpression*.
Value	Enter some expression that evaluates to Y or N, using the CfgEval() function to reference session values or procedural conditional variables.

3 On UI controls for this product (such as relationships, groups, products, tabs, attributes, linked items or resources) create UI properties with the name DynamicHide plus the name you gave to the expression in Step 2, and with a value that is Y or N.

a Select the relationship, group, product, tab, attribute, linked item or resource.

b Navigate to the Properties view.

c In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *DynamicHide* plus the name you gave to the expression. For example, you might enter *DynamicHide MyExpression*.
Value	Enter *Y* or *N*.

The control is hidden when the value of the expression matches the value you gave to this property (*Y* or *N*).

4 Release the product.

Example of Dynamically Showing and Hiding Siebel Configurator Controls

This topic gives one example of dynamic show and hide. You might use this feature differently, depending on your business model.

This topic shows how to create the example described earlier. A business sells tee-shirts that can be personalized with the name and business name of the purchaser or that can be bought without personalization. The root product is a tee-shirt with attributes including the following: Personalize and Name. If the user selects the Personalize checkbox (which sets the Personalize attribute to Y), then an additional Siebel Configurator tab is displayed that allows the user to enter the customer's name and the business name.

This example assumes that you have defined the product and have defined the Siebel Configurator UI, including the tab used for personalization. The requirement is that you use the dynamic show and hide feature to hide the Name attribute when the Personalize checkbox is not selected (so the Personalize attribute is N).

To dynamically show and hide a Siebel Configurator control

1 On the Tee-Shirt product create the CfgUIStateModel UI property:

a Navigate to the Administration - Product screen, Product Definitions view.

b Lock the Tee-Shirt product.

c Drill down on the Work Space version of the Tee-Shirt Product.

d Navigate to the Properties view.

e In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *CfgUIStateModel*.
Value	Enter *Y*.

2 On the Tee-Shirt product, create the second UI property:

■ In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *Expr1*.
Value	Enter *CfgEval($.[Teeshirt]#1.xa[Personalize].xf[Value])='No'*

3 On the Name attribute of the product, create the UI property:

a Select the Name attribute.

b Navigate to the Properties view.

c In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *DynamicHide Expr1*
Value	Enter *Y*

4 Release the root product.

Dynamically Making Siebel Configurator Controls Read-Only

You make Siebel Configurator controls read-only in the same way that you hide them. For more information, see "Dynamically Showing and Hiding Siebel Configurator Controls" on page 184.

The only difference is that, in Step 3 on page 185, you create a property for the control named DynamicReadOnly instead of DynamicHide.

Dynamically Displaying Images in Siebel Configurator

You can dynamically hide and show images in the Siebel Configurator user interface.

For example, the product is a tee-shirt that comes in two colors. It has an attribute named Color with the values Red and Blue. If user selects the attribute Red, then an image of the red tee-shirt is displayed. If user selects the attribute Blue, then an image of the blue tee-shirt is displayed.

You can dynamically display a number of images based on different conditions. That is, you can display Image1 when Expression1 is Y, display Image 2 when Expression2 is Y, and so on. For example, you can display dozens of images of tee-shirts with different colors.

Dynamic display of images is similar to dynamic display of Siebel Configurator controls. For more information, see "Dynamically Showing and Hiding Siebel Configurator Controls" on page 184. Differences are described in the following procedure.

NOTE: The DynamicImage property can be defined only on the root product and on relationships. It is not supported on component products.

To dynamically show and hide images in Siebel Configurator

1. Place a file for the image in the folder eapps\public\enu\images.
2. On the root product create the CfgUIStateModel UI property:
   a. Navigate to the Administration - Product screen, Product Definitions view.
   b. Lock the root product.
   c. Drill down on the Work Space version of the root product.
   d. Navigate to the Properties view.
   e. In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *CfgUIStateModel*.
Value	Enter *Y*.

3. On the root product create a UI property with the name NewImage Expr1 (where Expr1 is the name of the expression) and with a value that is the name of the image file:
   a. Navigate to the Administration - Product screen, Product Definitions view.
   b. Lock the product you want to change.
   c. Drill down on the Work Space version of the product you want to change.
   d. Navigate to the Properties view.

e In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *NewImage MyExpression* (where MyExpression is the name of the expression).
Value	Enter the name of the image file, including the name of the images directory. For example, enter *images/red_tshirt.jpg*

4 On the root product, create a UI property that defines the expression, with some given name, such as *MyExpression*, and with a value that is an expression that evaluates as Y or N:

- In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter a name for the expression, such as *MyExpression*.
Value	Enter some expression that evaluates to Y or N, using the CfgEval() function to reference session values or procedural conditional variables.

5 If you want to show and hide the image for the root product, add another property for the root product with the name DynamicImage plus the name you gave to the expression in Step 2, and with a value that is Y or N.

- In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *DynamicImage* plus the name you gave to the expression. For example, you might enter *DynamicImage MyExpression*.
Value	Enter *Y* or *N*.

The control is hidden when the value of the expression matches the value you gave to this property (*Y* or *N*).

6 If you want to show and hide the image at the relationship level, create this property for the relationship.

a Select the relationship.

b Navigate to the Properties view.

c In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Description
Name	Enter *DynamicImage* plus the name you gave to the expression. For example, you might enter *DynamicImage MyExpression*.
Value	Enter *Y* or *N*.

The control is hidden when the value of the expression matches the value you gave to this property (*Y* or *N*).

7 Release the product.

13 Siebel Configurator Web Templates

This chapter explains how to customize the Web templates that are used to create Siebel Configurator selection pages. It includes the following topics:

About Customizable Product Web Templates

Customizable Product Web templates are files that control all aspects of how selection pages display. You can customize the look and feel of selection pages by modifying these templates.

Customizable Product Web templates are stored in the installation subdirectory \webtempl. The templates that control display of selection pages begin with eCfg. There four types of Web templates used by Siebel Configurator.

- **Base Theme.** The base theme defines an HTML table-based page layout. This layout is used for all selection pages and provides the basic look and feel. The base theme calls the required product theme, which displays in one or more table cells created by the base theme.
- **Product Theme.** The product theme template defines how selection items and options are displayed. The one-page theme displays all the configurable items on one selection page. The tab themes display items on a series of tabbed pages. The wizard theme leads the user from one selection page to the next. Product themes also create table layouts that define how items display within the cells of the base theme. Product themes contain for-each loops that iterate through the customizable product and identify relationships, items, attributes, and so on. Product themes call group themes and control themes.

- **Group Theme.** If you use a tab or wizard theme, you determine what items appear on a page by defining groups. All the items in one group display on one page. A group theme template specifies how a group displays on a page.
- **UI control template.** UI control templates define what type of UI control is used for selecting items. You can choose from several types of check box, radio button, and text box controls. The control template iterates through each group, identifies all the items, and then creates a form that displays the items for selection. The forms display in the table cells created by the product theme.

Web templates are not themselves HTML files, but they do contain a combination of HTML table commands, JavaScript, and Siebel Web Engine (swe) commands. The swe commands are in XML format.

NOTE: Open UI Web Templates have been modified to remove Table, Row and Data tags. Open UI Web Templates have also had JavaScript functions moved to the renderer or model JavaScript library. Open UI Web Templates are also stored in the installation subdirectory webtempl\ouiwebtempl.

These commands are used in Web templates as follows:

- **HTML table commands.** These commands are used to define page layout. Control templates create tables and then create forms that display in these tables. Control template tables display in the cells created by product theme templates. Product theme templates displays in cells created by the base theme template.
- **JavaScript.** JavaScript commands are used to create arrays for storing data about the customizable product obtained from the UI service. They are also used in control templates to create forms.
- **swe:include.** This command specifies the name of a template to include. This command links the Web templates together. During iterative processing this command causes the Web Engine to dynamically retrieve, insert, and parse the included Web template.
- **swe:for-each.** This command provides iterative processing. It traverses the customizable product beginning at the product root and identifies specified parts for display. The CfgLoop Type specifies the type of items to be identified, such as group, relationship, or attribute.
- **swe:control.** This XML element defines what control type to display with an item. To define a UI property, you substitute a variable name for the value of the CfgFieldName attribute in this element. Inserting UI property variables in Web templates is how you customize the way an item displays.

Web templates are used by the Frame Code Engine to create HTML selection pages, as shown in Figure 12.

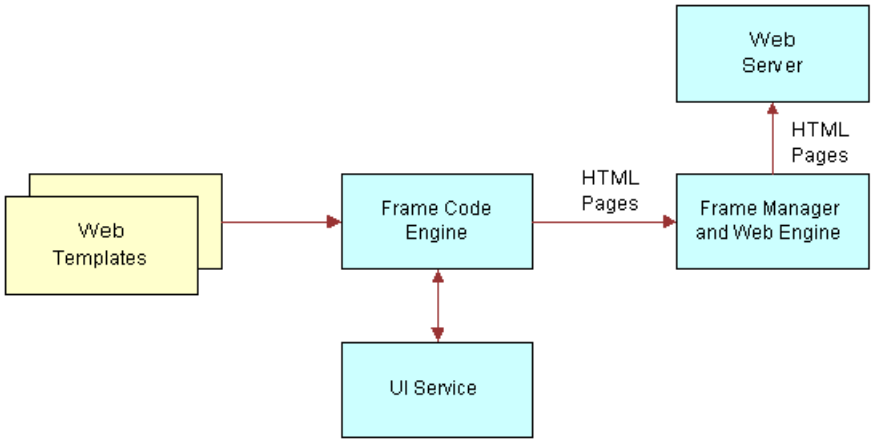

Figure 12. Web Template Processing

When a customizable product is called for display, an instance is created and stored in memory. The Frame Code Engine requests information about the product from the UI service and uses it to provide the values required in the Web templates. The Frame Code Engine then builds selection pages and forwards them to the Web Engine Frame Manager. The pages are then provided to the Web Server.

About UI Properties in Web Templates

The Properties view lets you customize the way items display in a configuration session. You do this by defining a UI property for the desired items. The UI property definition is a name-value pair where the variable name is one you have entered in the Web template that controls the display of the item. The value can be a string, HTML commands, XML commands, or JavaScript that defines what you want to display instead of the default item name.

The Properties view displays all the items in the customizable product. To change how Web templates display, you can define UI properties of the following items:

- Customizable Product Name (root product)
- Attribute
- Relationship
- Group
- Linked Item
- Resource.

For more information about using the Properties view, see Chapter 12, “Siebel Configurator UI Properties.”

The Web Engine uses for-each loops to iterate through each level of a customizable product. At each level, it determines what items occupy that level and what Web template to use for displaying the items. When you modify a Web template and assign it to group member, the template is used to control the display of all the group member's items. For example, if you assign a radio button control template to a relationship containing five items. The control template is used to define how each of the five items displays. Thus, if you have inserted a UI property variable in the radio button control template, then you must define a UI property for all five items.

About UI Property Values

The value you assign to a UI property name for an item can be text, HTML commands, or JavaScript commands. If the value includes HTML or JavaScript commands, it is important to test them for correctness before entering them in the Properties view.

If you do not test the commands and they have errors, this can prevent display of the selection pages. If the value is a text string that does not include commands, you do not need to test it.

You test the commands included in the value of the UI property name by inserting them in an HTML file and checking that they display correctly in a Web browser. Observe the following guidelines for including HTML commands or JavaScript in a UI property name value:

- Avoid using tags or tag attributes common only to Internet Explorer or to Netscape.
- Use DHTML commands with caution. Thoroughly test them before using them as the value of a UI property name.
- HTML statements must be self-contained and complete. Use opening and closing tags.
- Use table tags very carefully. Make sure the table you define is sized correctly for the space it will occupy.
- If you insert JavaScript using the <Script> tag, avoid statements that manipulate the document. Also avoid routines that rely on specific page content. If the content is not present, the script may fail and the page may not display.
- Do not use animated images or animated text.

NOTE: Siebel Open UI recommends that all JavaScript commands be done in the renderer or model library and all stylist language is done in the CSS library.

HTML Text Formatting Commands

You can use HTML text formatting commands to enhance the way an item name displays. Here are several examples:

- You can define a UI property value that adds formatting to the item name. For example, you want the item name Lamp to display in boldface. You would assign the following UI property value to the item Lamp: Lamp.

- You can add a message next to an item. If the message is lengthy consider creating a small, two-cell table. Put the item name in the first cell, and put the explanation in the adjacent cell. The value of the UI property name for the item would then be the HTML table commands, including the item name and message. The base theme and product theme Web templates use tables to layout the Web pages. This means the table you create for the item will be located within a cell of the table that contains the whole Web page. Carefully review the table structure of the base theme and product theme Web templates before creating tables for UI properties.

The following HTML tag types can be used as values for UI property names:

- Text markup tags (<b>, , and so on)
- Table tags
- Content presentation and flow tags (<address>, <nobr>, <plaintext>, and so on)
- Formatted list tags
- Rule, image, and multimedia tags (<img>, <map>, <marquee>)
- Forms tags (<button>, <input type> and so on.). You can use these tags to pass user input to JavaScript routines that are part of the UI property name value.
- Hyperlinks. You must include Target = ""in the link tag (<a>) definition. This causes the link to load in a new browser window. If the link loads in the session browser window, the user will have to click the Back button to return to the session. This can cause the session to lose its context and can cause Web Engine problems.

Do not use the following tag types in UI property name values:

- Header tags (<base>, <basefont>, and so on)
- Skeletal/Layout tags (<frameset>, <body>, and so on)

Images

Use the HTML <img> tag as the UI property value when you want to retrieve and display images. You can shorten the path specification for the src attribute by storing the images in the same directory as other images used by the Web Engine.

The Web Engine stores its images in the following installation subdirectory (Windows path syntax):

```
PUBLIC\<language>\IMAGES
```

The <language> variable is the three-letter language identifier for the language selected during installation. For example, if you selected English during the install, the Web Engine image files are located in the PUBLIC\enu\IMAGES subdirectory.

When you specify the src path in the <img> tag, you only need to specify the IMAGES directory and the file name. For example, you want to retrieve red.gif from the IMAGES directory and use it to replace the attribute name Red. In the Properties view, you would assign a UI property name to the Red attribute and specify the following value (Windows path syntax):

```
<img src="IMAGES\red.gif">
```

Before validating the UI design, you must test this value to make sure it behaves as expected in the browser. Here is an example of an HTML file for testing image retrieval (English language installation, Windows path syntax):

```
<html>
<head>
<base href="C:\installdir\PUBLIC\enu\">
</head>
<body>
<img src="IMAGES\red.gif">
</body>
</html>
```

Add HEIGHT and WIDTH attributes to the <IMG> tag to make the image the correct size. Consider making the image somewhat smaller than needed and then increasing its size when you validate the UI design. This prevents the image from causing page layout problems when you first validate it.

Creating a New Web Template

The most common reason for creating a new Web Template is to insert UI property variables in the template to change how an item in a customizable product displays. The best way to create a new Web template is to modify an existing template and save it to a new file name.

The process for setting up a new template has two steps:

a. Create a new Web template.

b. Add the new template to the appropriate dialog box

NOTE: For more information about modifying and providing new Web templates for Open UI, see *Configuring Siebel Open UI*.

To create a new Web template

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired product.
3. In the Versions list, click the Work Space version.
4. Open the desired dialog box and locate the desired template.

   For example, if you want to create a new base theme template, navigate to the User Interface view and click the select button in the Base Theme field to open the dialog box.
5. Write down the filename of the template you want to modify.

   The filename is shown in the Template field. The filename ends in .swt.
6. Open the Web template in a text editor, modify it as desired, and save it to a new filename.

   In most cases, you will insert a UI property variable name into the template.

The next step is to add the new template to the dialog box.

To add a Web template to the dialog box

1 Open the dialog box again.

This is the dialog box you used to determine the filename of the template.

2 In the dialog box, click New and fill out the form for adding a new template and click OK.

- **Name:** Enter a descriptive name. For example: Modified Base Theme.
- **Template:** Enter the filename of the template you created.
- **Description (Optional):** Enter a brief description of the template.

This adds the new template to the dialog box. You can now assign the new template to an item in the customizable product. For example, if you created a new base theme template, you can assign this template to the customizable product.

Modifying the Display Name of a Customizable Product

This topic explains how to modify the customizable product name in the base theme. The base theme defines the basic page container within which the product theme displays. The base theme includes a header that contains the customizable product name that is stored in the Siebel database. You can define a UI property that changes the name of the customizable product or adds artwork or other formatting.

To change the name of the customizable product in a base theme, you insert a variable in the base theme Web template. This variable tells the Web Engine to get the item name from a defined UI property rather than using the customizable product name.

To change the display name of the customizable product, perform the following tasks:

1 “Creating a New Base Theme Template” on page 197. You do this by saving a copy of the base theme template. Then you insert a variable name in the new template.

2 “Assigning the New Base Theme Template” on page 198. Assign the new base theme template to the customizable product.

3 “Defining a UI Property for the Customizable Product” on page 199. This value is what displays in the selection pages.

Creating a New Base Theme Template

You must create a new Web template to customize the display name of the customizable product. You do this by copying an existing base theme template. Then you insert a UI property variable into the copy. Finally, you assign the new Web template as the base theme for the customizable product

To insert a UI property variable into a base theme Web template, you must locate the swe:control element that governs the display of the customizable product name. The base themes have the same basic layout:

- The first <Table> tag creates the page container. Additional <Table> tags partition the page vertically.
- There are no swe:for-each loops in the template.
- Near the start of the file, refer to the first <table> tag. Within the table definition, locate the “Product Title” comment. After the comment locate the first swe:control element.
- One of the attributes of this element is CfgFieldName = “CxObjName”.
- Replace CxObjName with the name of the UI Property variable. The name must be preceded with a period (.). For example: “.NewProductName”.

To create a new base theme template

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired customizable product.
3. In the Versions list, click the Work Space version.
4. In the User Interface view, click the select button in the Base Theme field and open the dialog box.
5. Write down the filename of the base theme.

   The filename is shown in the Template field. The filename ends in .swt.
6. Open the Web template in a text editor and save it to a new filename.

   The Web templates are located in <install-dir>\webtempl (NT path syntax), where <install-dir> is the path to the Siebel installation directory. The new filename must end in .swt.
7. In the new template, locate the swe.control element containing the CfgFieldName= “CxObjName” attribute.
8. Replace “CxObjName” with a UI property variable name and save the file.

   The variable name must begin with a period (.). For example: “.NewProductName”. Verify that the variable name is enclosed in quotes.

Assigning the New Base Theme Template

To assign the new base theme template to the customizable product, you first add the template to the Pick UI Style dialog box. Then you select it as the base theme.

To assign the new base theme template to the customizable product

1. In the customizable product Work Space version, click the select button in the Base Theme field and open the dialog box.
2. In the dialog box, add a new record and complete the necessary fields:
   - **Name.** Enter a descriptive name. For example: Modified Base Theme.
   - **Template.** Enter the filename of the template you created.

- **Description (Optional).** Enter a brief description of the template.

3 In the Pick UI Style dialog box, click the template you added and click OK.

This assigns the new base theme template to the customizable product.

Defining a UI Property for the Customizable Product

The last step is to define a UI property for all the customizable product.

To define a UI property for the customizable product

1 Navigate to the Properties view for the customizable product.

2 In the explorer applet displaying the contents of the customizable product, click the name of the customizable product. It is the first item listed.

3 In Item Display Properties, add a new record and complete the necessary fields:

- **Name.** Enter the name of the UI property variable you entered in the base theme template. Do not include the period (.) that begins the name.
- **Value.** Enter the customizable product name you want to display. Include any HTML formatting needed.

Example of Modifying the Display Name of a Customizable Product

You have a customizable product called Premier Workstation. You want to display your company logo beside the product name in the base theme. The logo filename is logo1.gif. You have placed the file in the Siebel installation subdirectory \PUBIC\enu\IMAGES (NT path syntax, English language installation).

Use the User Interface view to create the Web pages for configuring the customizable product. Validate the customizable product, and verify that the pages display correctly.

To create a new base theme Web template and assign it to the customizable product

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Products list, select and lock the desired customizable product.

3 In the Versions list, click the Work Space version.

4 In the User Interface view, click the select button in the Base Theme field and open the dialog box.

5 Write down the filename of the base theme.

6 Open the base theme Web template and save it to a new file name: eCfgNewProductTheme.swt

7 Open the new template and locate the swe.control element. Set CfgFieldName=“.NewProductName”. The first character in the variable name must be a period (.). The variable name must be surrounded by quotes. Save the file.

8 In the customizable product Work Space record, click the select button in the Base Theme field and open the dialog box.

9 Click Add and add the new template. Then, select it as the base theme template.

To define a UI property for the customizable product

1 Navigate to the Properties view and select the customizable product.

2 Define a UI property for it as follows:

- **Name.** NewProductName. Do not put a period before the name. This is the variable name you inserted in the template file.
- **Value.** <img src=”\images\logi1.gif”> Premier Workstation.

Modifying the Display Name of Groups

This topic explains how to modify group names that display in the product themes. For example, you can use UI property definitions to define what displays on each tab in a tabbed product theme.

To change the name of a group in a product theme, you insert a variable in the product theme Web template. This variable tells the Web Engine to get the item name from a defined UI property rather than using the group name you defined in the User Interface view.

To change the display name of a group, perform the following tasks:

1 “Creating a New Product Theme Template” on page 200. You do this by saving a copy of the product theme template. Then you insert a variable name in the new template.

2 “Assign the New Product Theme Template” on page 201. Assign the new product theme template to the customizable product.

3 “Define a UI Property for all the Groups” on page 202. For each group name, you give the variable the desired value. This value is what displays in the configuration Web pages.

You must define a UI property for all the group names, not just the ones you want to change. This is because the product theme template determines the display name of the all the groups. When you insert a UI property variable in the template, the Web Engine gets all the group names from UI property definitions.

Creating a New Product Theme Template

You must create a new Web template to customize the display of group names. You do this by copying an existing product theme template. Then you insert a UI property variable into the copy. Finally, you assign the new Web template as the product theme for the customizable product.

To insert a UI property variable into a product theme Web template, you must locate the swe:control element that governs the display of the group name:

- **Tab Theme.** Locate the first swe:case tag. It is located near the start of the file.
- **Single Page Theme.** Locate the third <table> tag. It is within the for-each loop near the start of the file.
- **Wizard Theme.** Locate the first for-each loop. It is near the start of the file.
- Beneath the location specified previously, locate the first swe:control element. It defines the group data record. One of the attributes of this element is CfgFieldName = “CxGroupName”.
- Replace CxGroupName with the name of the UI Property variable. The name must be preceded with a period (.). For example: “.NewGroupName”.

To create a new product theme template

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Product list, select and lock the desired customizable product.
3. In the Versions list, click the Work Space version.
4. In the User Interface view, click the select button in the Product Theme field and open the dialog box.
5. Write down the filename of the product theme.

   The filename is shown in the Template field. The filename ends in .swt.
6. Open the Web template and save it to a new filename.

   The Web templates are located in <install-dir>\webtempl (NT path syntax), where <install-dir> is the path to the Siebel installation directory. The new filename must end in .swt.
7. In the new template, locate the swe.control element containing the CfgFieldName= “CxGroupName” attribute.
8. Replace “CxGroupName” with a UI property variable name and save the file.

   The variable name must begin with a period (.). For example: “.NewGroupName”. Verify that the variable name is enclosed in quotes.

Assign the New Product Theme Template

To assign the new product theme template to the customizable product, you first add the template to the Pick UI Style dialog box. Then you select it as the product theme.

To assign the new product theme template to the customizable product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Product list, select and lock the desired customizable product.
3. In the Versions list, click the Work Space version.
4. In the User Interface view, click the select button in the Product Theme field and open the dialog box.

5 In the dialog box, click New and complete the necessary fields:

- **Name.** Enter a descriptive name. For example: Modified Tab Theme.
- **Template.** Enter the filename of the template you created.
- **Description (Optional).** Enter a brief description of the template.

6 In the Pick UI Style dialog box, click the template you added and click OK.

This assigns the new product theme template to the customizable product.

Define a UI Property for all the Groups

The last step is to define a UI property for all the groups.

To define a UI Property for all the groups

1 Click the Properties view tab.

2 In the box displaying the contents of the customizable product, select the first group in the Group section.

3 In Item Display Properties, add a new record and complete the necessary fields:

- **Name.** Enter the name of the UI property variable you entered in the product theme template. Do not include the period (.) that begins the name.
- **Value.** Enter the group name you want to display. Include any HTML formatting needed.

4 Save the new record.

5 Perform these steps for all the groups.

6 Open the Item Display Properties menu and click Validate.

This starts a configuration session. Verify that the group names display correctly.

Example of Modifying the Display Name of Groups

You have a customizable product that has three groups: Group A, Group B, and Collection C. You want to change the name of Collection C to Group C in the selection pages.

Use the User Interface view to create the Web pages for configuring the customizable product. Validate the customizable product, and verify that the pages display correctly.

To create a new product theme template and assign it to the customizable product

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Product list, select and lock the desired customizable product.

3 In the Versions list, click the Work Space version.

4 In the User Interface view, click the select button in the Product Theme field and open the dialog box.

5 Write down the filename of the product theme.

6 Open the product theme Web template and save it to a new file name: eCfgModifiedTabTheme.swt

7 Open the new template and locate the swe.control element. Set CfgFieldName=".NewGroupName". The first character in the variable name must be a period (.). The variable name must be surrounded by quotes. Save the file.

8 In the customizable product Work Space record, click the select button in the Product Theme field and open the dialog box.

9 Add the new template.

10 Select the template as the product theme template.

To define a UI property for each of the groups

1 Click the Properties view tab and select Collection C in the Group section.

2 Define a UI property for it as follows:

- **Name.** NewGroupName. Do not put a period before the name. This is the variable name you inserted in the template file.
- **Value.** Group C.

3 Select Group A.

4 Define the UI property for it as follows:

- **Name.** NewGroupName. Do not put a period before the name.
- **Value.** Group A.

5 Select Group B.

6 Define the UI property for it as follows:

- **Name.** NewGroupName. Do not put a period before the name.
- **Value.** Group B.

7 Open the Item Display Properties menu and click Validate.

This starts a configuration session. Verify that the group names display correctly.

Modifying the Display Name of Items

This topic explains how to modify the display name of an item, such as an attribute, relationship, linked item, or resource. A common reason for doing this is that you want the item name that the customer sees to be different than the name you have in the product table for a specific version of the customizable product.

For example, a computer component is called 256 MB disk drive in the product table. However, in an upcoming offering, you want to call this item the Standard 256 MB Disk. Rather than change the item name in the product table, you can change it using the Properties view. This change is specific to the customizable product and is stored with it when you release the product. You can use the Properties view to change item names with each release of a customizable product, without having to change the item name in the product table.

To change the display name of an item, you insert a variable in the UI control Web template that governs display of the item. This variable tells the Web Engine to get the item name from a defined UI property.

To change the display name of a relationship item in the form, perform the following tasks:

1. "Creating a New UI Control Template" on page 204. You do this by saving a copy of the UI control template you selected in the User Interface view. Then you insert a variable name in the new template.
2. "Assigning the New UI Control Template" on page 205. Assign the new template to the item.
3. "Defining a UI Property for the Item" on page 206. Define a UI Property for the item.

Creating a New UI Control Template

You must create a new UI control template to customize the display of an item. You do this by copying an existing template. Then you insert a UI property name variable into the copy. Finally, you assign the new template as the UI control template for the item.

To insert a UI property variable into a Web template, you must locate the swe.control element that governs the display of the item. The UI control Web templates all have the same basic layout:

- A swe.include statement reads in a header file. This file places the relationship name at the start of the form that will contain the items.
- A swe for-each loop iterates through the relationship in the customizable product and loads its items into an array.
- A second swe for-each loop reads the array and constructs the form.
- A variable called DisplayValue near the beginning of the second swe for-each loop defines what item name appears next to each instance of the control in the form.
- The DisplayValue variable is set equal to a swe.control element that contains an attribute CfgFieldName= "CxObjName". Replace CxObjName with the name of the UI property variable. The name must be preceded with a period (.). For example: ".NewName".

To create a new UI control template

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Product list, select and lock the desired customizable product.
3. In the Versions list, click the Work Space version.
4. Click the User Interface view tab.

5 Click the group name containing the item whose display you want to modify.

The items belonging to the group you select display in the Group Item List.

6 In the Group Item List, select the item you want to modify, for example a relationship.

7 In the record you selected, click the select button in the UI control field.

The Pick UI Style dialog box displays. The control you have selected for the group member is highlighted.

8 Write down the filename of the Web template that governs the display of this control.

The filename is shown in the Template field. The filename ends in .swt.

9 Open the Web template in a text editor and save it to a new filename.

The Web templates are located in <install-dir>\webtempl (NT path syntax) where <install-dir> is the path to the Siebel installation directory. The filename must begin with eCfg and end with .swt. For example: eCfgport_modifiedcheckbox.swt

10 In the new template, locate the correct swe.control element containing the CfgFieldName= "CxObjName" attribute.

11 Replace "CxObjName" with a UI property variable name and save the file.

The variable name must begin with a period (.). For example: ".NewName". Verify the variable name is enclosed in quotes.

Assigning the New UI Control Template

To assign the new Web template to a group item, you first add the template to the Pick UI Style dialog box. Then you select it as the template for an item in the group.

To assign the new UI control template

1 In the User Interface view Group List, click the group name containing the item whose display you want to modify.

The items belonging to the group you select display in the Group Item List.

2 In the Group Item List, click the item you want to modify.

This selects the record.

3 In the record you selected, click the select button in the UI control field.

The Pick UI Style dialog box appears.

4 In the dialog box, click New and complete the necessary fields:

- **Name.** Enter a descriptive name. For example: Modified Check Box.
- **Template.** Enter the filename of the template you created.
- **Description (Optional).** Enter a brief description of the template.

5 In the Pick UI Style dialog box, click the template you added and click OK.

This assigns the new template to the item and closes the dialog box.

6 In Group Item List, save the revised record.

Defining a UI Property for the Item

The last step is to assign the variable in the new Web template to the item. Then you enter a value for the variable. The value you enter is what displays in a configuration session.

For example, if you assigned the template to a relationship, you must define a UI property for each item in the relationship.

To define a UI Property for the item

1 Click the Properties view tab.

2 In the box displaying the contents of the customizable product, select the item on which you want to define the UI property.

3 In Item Display Properties, add a new record and complete the necessary fields:

- **Name.** Enter the name of the UI property variable. Do not include the period (.) that begins the name.
- **Value.** Enter the value you want to the variable to have for this item.

4 Save the new record.

5 Open the Item Display Properties menu and click Validate.

This starts a configuration session. Verify that the item names display correctly.

Example of Modifying the Display Name of Items

You have a customizable product that includes a relationship called Hard Drives. This relationship contains several disk drives. Along with the name of each drive, you want to display a picture of the drive. You want to assign radio buttons as the UI control for selecting the drives.

To prepare for defining the UI Property

1 Use the User Interface view to create the selection pages for configuring the customizable product.

2 Validate the customizable product, and verify that the pages display correctly.

3 Create a .gif or .jpg for each hard drive.

4 Test each file by displaying it in a browser. Use the HTML <img> tag to set the exact size of each image. The <img> tag will be the value you assign to the UI Property variable.

To create a new UI control template and assign it to the Hard Drives relationship

1. In the User Interface view, select the group containing the Hard Drives relationship.
2. In Group Item List, select the Hard Drives relationship.
3. Open the Pick UI Style dialog box and write down the name of the radio button template assigned to the relationship.
4. Open the radio button template and save it to a new file name: eCfgModifiedRadioButton.swt
5. Open the new template and locate the correct swe.control element.
6. Set CfgFieldName= ".NewName".

   The first character in the variable name must be a period (.). The variable name must be surrounded by quotes.
7. Save the file.
8. In the User Interface view Group Item List, select the relationship containing the hard drives.
9. Open the Pick UI Style dialog box and click Add to add the new template.
10. Select the new template as the control template for the Hard Drives relationship.

To define a UI property for each of the Hard Drives

1. Navigate to the Properties view and select the first drive in the Hard Drives relationship.
2. Define a UI property for it as follows:
   - **Name.** NewName. Do not put a period before the name. This is the variable name you inserted in the template file.
   - **Value.** Enter the HTML syntax for displaying the drive name and retrieving the image.
3. Repeat these steps to define a UI property for each hard drive in the relationship.
4. Open the Item Display Properties menu and click Validate.

   This starts a configuration session. Verify that the item names display correctly.

Siebel Configurator Web Templates for Open UI

Siebel Product Configurator in Open UI provides a set of Web templates that are enabled for responsive Web design. These templates provide the user with two panes, giving the user the choice of using a tab or wizard approach to customizing products.

As delivered, these Web templates must be used together and cannot be used with other Siebel Product Configurator Web templates.

These templates are described in Table 15.

Table 15. Open UI Web Templates for Siebel Product Configurator

File Name	Name	Description
ecfgproducttabs_v14.swt	Tab Product Theme OUI	The product tab theme displays the product configuration UI with tabs and with Previous and Next buttons to navigate between groups.
ecfggroupstandard_v14.swt	Standard Group Theme OUI	The group theme displays a div section (which is like a table row) for each relationship that is part of the group definition.
ecfgcontrolcomboprice_v14.swt	Combo Box with Price OUI	The control combo price theme displays the products that are part of the relationship as option items in a combo box.
sisomecfgbaseautorepricetwocol_v14.swt	SIS OM base Theme With Auto Pricing OUI	The base theme displays two parts in the configuration page. The first section displays the product details, pricing and any messages associated with the product configuration, and the second section displays the product configuration UI.
ecfgcontrolradioprice_v14.swt	Radio Buttons with Price OUI	The control radio price theme displays the products that are part of the relationship as option buttons.
ecfgcontrolcheckprice_v14.swt	Check Box with Price OUI	The control check price displays the products that are part of the relationship as check box options.
ecfgobjmessages_v14.swt	n/a	Called as include SWT. This template controls the product related information that is displayed in the second section.
ecfgportheader_v14.swt	n/a	Called as include SWT. This template controls the header for the div section displayed for the relationship.
ecfgportfooter_v14.swt	n/a	Called as include SWT. This template controls the footer for the div section displayed for the relationship.

Table 15. Open UI Web Templates for Siebel Product Configurator

File Name	Name	Description
ecfgtoplevelbuttons_v14.swt	n/a	Called as include SWT. This template displays various action buttons that control the product configuration.
ecfgobjglobalsignals_v14.swt	n/a	Called as include SWT. This template displays the various messages that are generated during the configuration process.

About Customizing Configurator Templates for Open UI

The Open UI framework for Configurator uses the following architecture, as you can see in Figure 13.

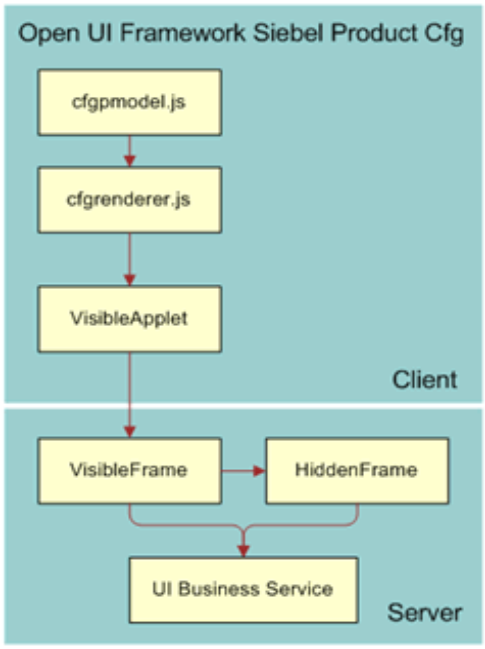

Figure 13. Siebel Product Configurator Architecture

The Siebel Web Templates function as follows:

- Open UI uses <DIV> and <CLASS> tags to replace the <TABLE>, <TR>, and <TD> tags UI attributes.

- Open UI removes the inline scripts embedded with <SCRIPT> tags.

In the Object Manager (written in C++), Open UI functions as follows:

- The applet (Cfg Cx Runtime Instance Frame (JS HI)) is the only point of communication. All requests from the client (browser) are delegated through it.
- There is no longer server generated JavaScript. Necessary data is packaged into a payload so that the client model renderer can process it and generate proper scripts.

In the client (browser), Open UI functions as follows:

- ConfiguratorPM (cfgpmodel.js) is extended using the PresentationModel (pmodel.js) architecture of Open UI. It maintains the product instance changes and handles the communication between the UI and the Server.
- ConfiguratorPR (cfgrenderer.js) is extended from PhysicalRenderer (phyrenderer.js) architecture of Open UI. It handles the rendering and evaluating of UI elements from ConfiguratorPM.

Customizations of Web Templates must use libraries that have been extended to accommodate the extended PresentationModel or Renderer. These extended libraries are also compliant with Open UI's architecture.

All customizations are placed in extended libraries for easier upgrades.

14 Configuration Constraints

This chapter explains how to create configuration constraints. It also explains how to create constraint templates. It includes the following topics:

About Configuration Constraints

A configuration constraint defines how two items in a customizable product or a product with attributes are related. For example:

- In a product with components, Component A and Component B are mutually exclusive. If the user picks one, then you want to prevent them from picking the other. One way you can do this is by writing a configuration constraint: Component A excludes Component B. The Constraints view provides a constraint template to help you write this constraint.
- A product with attributes comes in a number of different colors and sizes. However, the color red is not available in the XL size. One way you can prevent users from picking a product with the color red and the XL size is to write a configuration constraint: Red excludes XL.

Constraint templates are constraint statements where you replace variables in the statement to create a configuration constraint. The Constraints view provides constraint templates for the most common types of configuration constraints. You can also create your own constraint templates.

In the Constraints view, you create a configuration constraint by first selecting a constraint template. Then you pick items from the customizable product and operators or even other constraint templates to replace the variables in the constraint. Both arithmetic and logical operators are provided by the Constraints view.

Configuration constraints you create apply only to the current product, and are stored as part of it. In contrast, constraint templates reside in the Constraints view and can be used with any product.

For information about the templates that are available with the product, see Chapter 17, "Configuration Constraint Template Reference."

About the Advanced Constraint Configurator

The Advanced Constraint Configurator is a new constraint library for Siebel Product Configurator that can be utilized to apply existing, or new, customizable product constraints.

The Advanced Constraint Engine is not available by default, therefore you must actively install Enterprise Cache and Advanced Constraint Engine using the Siebel Enterprise Server installer. If you are installing Siebel for the first time, then you can include Enterprise Cache and Advanced Constraint Engine with other Siebel Enterprise Server components, such as the Siebel Server. However, if you perform a migration installation, then you must install the other Siebel Enterprise Server components first, for example, the database, gateway, and so on, and then install Enterprise Cache and Advanced Constraint Engine in a separate location.

The following considerations apply to installing the Advanced Constraint Engine:

- You can install Enterprise Cache and Advanced Constraint Engine separately.
- You configure and deploy the Advanced Constraint Engine using Siebel Management Console. You must configure and deploy profiles for the main Siebel server modules, and for Enterprise Cache, both the server and the client, before you can configure and deploy Advanced Constraint Engine.

For more information, see *Siebel Installation Guide* for the operating system you are using.

Process of Setting Up Advanced Constraint Configurator

To set up Advanced Constraint Configurator, perform the following tasks:

1. "Activating the Cfg Ora Engine Runtime Interaction Workflow" on page 213
2. "Setting Up Advanced Constraint Configurator System Preferences" on page 213
3. "Setting Up the EnableOraCfgEngine UI Property" on page 214

Activating the Cfg Ora Engine Runtime Interaction Workflow

After installing the Advanced Constraint Configurator, activate the Cfg Ora Engine Runtime Interaction workflow. For more information on activating a single workflow process, see *Siebel Business Process Framework: Workflow Guide*.

Setting Up Advanced Constraint Configurator System Preferences

Perform the steps in the following procedure to update up the Advanced Constraint Configurator system preferences.

To set up Advanced Constraint Configurator system preferences

1. Log in to a Siebel application on the Siebel Server.
2. Navigate to Administration - Application, and then the System Preferences view.
3. In the System Preferences list, either query for the following system preferences, or create the system preferences if they do not already exist, then enter the values shown in the following table:

System Preference	System Preference Value	Description
ConstraintEngineHost	<host>:<TLS-port>	Specifies the host and TLS port numbers where the constraint engine is deployed to access the Advanced Constraint Configurator. **NOTE:** This applies only to products using Oracle's Advanced Constraint Configurator.
EnableOraCfgEngine	N/Y	Enables the Advanced Constraint Configurator at the enterprise level. This value is overridden by the UI property found at the Customizable ProductThe default value for this system preference is N.

Setting Up the EnableOraCfgEngine UI Property

Create the UI property shown in the following table:

UI Property	UI Property Values	Description
EnableOraCfgEngine	Y or N	Allows the controlled deployment of the ACT library by activating the ACT library for a customizable product. **NOTE:** This UI property overrides the EnableOraCfgEngine system preference.

For more information on defining a UI property, see "Defining a UI Property" on page 182.

About Start and End Dates for Configuration Constraints

For each constraint you create, you can set effective dates that control when the constraint is active. You can set both a start date and an end date. On the start date the constraint is used to compute all solutions presented to the user when they configure a product. On the end date, the constraint is no longer used to when computing solutions.

Specifying start and end dates in combination has the following effects:

- **Both a start and end date specified.** The constraint becomes active on the start date and becomes inactive on the end date.
- **Start date specified.** The constraint becomes active on the start date and remains active thereafter.
- **End date specified.** The constraint is active when the version is released for use, and becomes inactive on the end date.

The start date is determined using the Siebel server's system date. The start and end dates work as follows in relation to the date the product is released (release date):

- If the release date is earlier than the start date, the constraint becomes active on the start date.
- If the release date is later than the start date, the constraint is active when the product is released.
- If the release date is earlier than the end date, the constraint becomes inactive on the end date.
- If the release date is later than the end date, the constraint is inactive when the product is released, and the constraint remains inactive.

When you are validating a product, you can temporarily activate or deactivate constraints in the current work space by clicking the constraint's Active box in the Constraints List. This lets you simulate how the constraint will behave on the start and end date.

For example, you can test a constraint with a start date in Validate mode using the following process:

1. Deselect the constraint's Active checkbox to deactivate the constraint. Then go to Validate mode and test the product. This simulates what users will see before the start date when the constraint is not being used to compute solutions.
2. Select the constraint's Active box to activate the constraint. Then go to Validate mode and test the product. This simulates what users will see after the start date when the constraint is being used compute solutions.

You can specify start and end dates for constraints when you create them or by editing constraints after they have been created.

About the Configuration Constraints View

The Constraints view has three parts:

- "Constraint Listing"
- "Constraint Template Listing"
- "Constraint Statement"

Constraint Listing

When you go to the Constraints view, all the constraints defined for a product are listed. You can edit, copy, and delete the constraints in the listing.

Constraint Template Listing

When you click New Constraint or New Template in the constraint listing, the constraint template listing appears. This listing contains the pre-defined constraint templates in the Constraints view. It also lists any templates you have created.

The constraint templates provide the basic constraint types you need for creating configuration constraints. For example, there are constraint templates for exclude constraints, others for require constraints, and so on.

Each constraint template contains variables that you replace to create a configuration constraint. You can replace the variables with items from the customizable product, links, resources, expressions, or other templates.

Constraint Statement

When you select a constraint in the constraint template listing and click Continue, the Constraint Statement form displays. It contains the constraint template you selected. You build a configuration constraint by replacing the variables in the statement with items from the customizable product, with resources or links, with operators, or with other constraints. To move to another variable in the constraint statement, click it. The currently selected variable in the constraint statement displays with square brackets around it. Variable that are not current but can be selected, display an underline when the cursor is placed on them. When you select an item for a variable, it displays in red.

The items you can replace a variable with are grouped in the “Insert a” tab, located after the Constraint Statement. When you move between variables in the constraint, the groupings change to reflect your allowable choices.

In some templates when you replace a variable with a value, typically an expression, the Compound button becomes active. The Compound button lets you nest expressions within expressions. For example, you could use the Compound button to add two variables together where the second variable is itself an expression that adds two variables.

Overview of Using the Constraints view

When you create constraints, you use the constraints view in the following way:

1. In the constraint template listing, select the desired constraint template.
2. Select the first variable in the constraint.
3. Click the desired item grouping.
4. Pick the product, operator, or constraint template you want to insert.
5. Move to the next variable and insert the desired item.
6. When you are finished, save the constraint.

Guidelines for Creating Configuration Constraints

Consider the following guidelines when creating constraints:

- If constraints are needed, create at least one constraint early in the process of building a product. The presence of a configuration constraint, even if it is inactive, causes Siebel Configurator to check the product for errors more rigorously when you go to validate mode.
- Avoid writing constraints that use large quantities until you have verified the logic of the constraint. For example, write a constraint that refers 10 items and check it before changing the constraint to refer to 10,000 items. This prevents needless solution searches if the basic logic of the constraint is incorrect.
- Test each constraint after you create it. Consider inactivating constraints that are unrelated to the new constraint to facilitate troubleshooting. Test constraints by starting a configuration session and selecting the affected items. To start a configuration session, from the Constraints List menu, select Validate.

- If you are using asset-based ordering, you can minimize order problems if you avoid creating require rules that add items that are not tracked as assets to a customizable product. For example, you write a require rule that adds a one-time charge for Installation to a customizable product. You do not set the Track as Asset flag for Installation in its product record. This means Installation does not display as a customer asset. Then the customer requests an addition to the service. The call center agent selects the service, clicks Modify, and starts a configuration session. The Siebel Configurator engine adds Installation, because it is required by configuration rules. Installation is transferred to the quote even though it is not required by the service modification.

Creating Configuration Constraints

You can create configuration constraints for customizable products or for product classes.

If you create the constraints for product classes, they are inherited by products with attributes that are in this class and by products with components that are in this class.

CAUTION: The maximum size for a configuration constraint is 900 characters. Do not enter a constraint longer than this.

To navigate to the constraints view for products with components

1. Navigate to the Administration - Products screen, then the Product Definitions view.
2. In the Product list, select and lock the desired product.
3. In the Versions list, click the desired version.
4. Navigate to the Constraints view.

   The Constraints List appears with all the constraints that have been created for this product.

To navigate to the constraints view for product classes

1. Navigate to the Administration - Products screen, then the Products Classes view.
2. In the Product Classes list, select and lock the desired product class.
3. In the Versions list, click the desired version.
4. Navigate to the Constraints view.

   The Constraints List appears with all the constraints that have been created for this product.

To create a configuration constraint

1. Navigate to the Constraints view, as described previously.
2. In the Constraints List, click New Constraint.

   The Pick a Constraint list appears and lists the constraint templates available for creating constraints. The Constraint Statement form displays the syntax of the currently-selected constraint.

3 In the Pick a Constraint list, select the constraint template that you want, and click Continue.

NOTE: For descriptions of the constraint templates, see Chapter 17, "Configuration Constraint Template Reference."

The Constraint Statement form and "Insert a" list appear. The Constraint Statement tab contains the constraint template you selected. The "Insert a" tab lists the item groups available for the currently-selected variable in the constraint.

To return to the display of all the Web templates, click Back. To exit and return to the Constraints List, click Cancel.

4 In the Constraint Statement form, click the first variable you want to work on.

Variables are enclosed in square brackets. When you click a variable, it turns red to indicate it is selected.

5 In the "Insert a" list, select the item grouping containing the item you want to insert. In the dialog box, choose the desired item.

When selecting products, click the product's select button. If you click the product name, the dialog box displays product information.

The variable in the constraint template is replaced by the item.

6 Repeat these steps for each variable until you have built the desired constraint.

7 Click Save Constraint to save the constraint.

The Save button becomes active when you have selected values for all the variables in the constraint. Clicking the Save button causes the Save Constraint form to appear.

8 Fill out the fields in the Save Constraint form, and then click Save. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the constraint. You must use names that help you to locate the constraint using the Find button. For example, consider including the constraint type (excludes, requires and so on) in the constraint name, so you can search the Name field to find groups of constraints having the same constraint type, for example, all the exclude constraints.
Explanation	Enter an explanation of how the constraint works. You must enter explanations that help you to locate the constraint using the Find button. For example, consider including information that uniquely identifies the constraint, such as item names, so you can search the Name and Explanation fields to find a specific constraint.
Rule Statement	Displays the constraint statement that you built. To edit the constraint, click Edit.
Start Date	Optionally, specify a start date on which the constraint becomes effective.
End Date	Optionally, specify an end date after which a constraint becomes inactive. For more information about Start Date and End Date, see "About Start and End Dates for Configuration Constraints" on page 214.
Active	Select this checkbox to activate the constraint, so it is used to compute solutions. Use this feature in the current work space to simulate the behavior of constraints that will have a start date, end date, or both when you release the product. You can also use this feature to deactivate a constraint but retain it in a released version of the product. For more information, see "Activating and Deactivating Configuration Constraints" on page 223.

The constraint displays in the Constraints List.

9 From the Constraints List menu, select Validate.

This starts a configuration session. Verify that the constraint works correctly.

Tuning Configuration Constraints

The Siebel Configurator engine evaluates solutions by traversing the attributes and the relationships of the root customizable product, for example, the product the user adds to the Order or Quote. The engine implements a recursive algorithm to derive at a solution and uses the algorithm to resolve the deepest node of each branch of the hierarchy. Once the deepest node of a given branch has been resolved, the algorithm returns to the previous level of the hierarchy and resolves the next product to the deepest node. This calculation continues until all branches, and all products of the branch, have been resolved.

During this process of branch resolution, the attributes are also resolved. This default evaluation order might result in unnecessary calculation and result in performance issues for some customers. Hence Siebel Configurator allows the administrator to adjust the evaluation order for a customizable product by specifying the UI property, GoalMode, under the class or product.

When you define the GoalMode UI property for a class, the entire set of attributes and relationships that define the class is impacted. Classes might be defined with just attribute definitions, or they might be defined with product, attribute, and rules definitions, known as Product Classes. For more information about Product Classes, see "About Product Classes" on page 60.

Every member of the class, including sub classes, inherits the GoalMode UI property. If a member of a sub class also has a GoalMode UI property defined, this overrides the parent class property.

If you define the GoalMode UI property for a customizable product, only the first level attributes and relationships that define the Product Class is impacted. In this instance, the GoalMode UI property does not propagate down through the hierarchy of a customizable product. For example, when a relationship contains another customizable product, the Siebel Configurator logic looks for a new GoalMode UI property at the level provided by either the product itself, or at the level provided by the class of the product. If a GoalMode UI property is not defined, Siebel Configurator uses the default value of 0.

The supported values for the GoalMode UI property are described in the following table:

Value	Description
0	Evaluate relationships first, then attributes (default current behavior).
1	Evaluate attributes first, then relationships (reverse of default).
2	Mixed mode, evaluates relationships and attributes together as a single set.

Sequence Number List Column

The Sequence Number List Column is used to decide the order of evaluation for both attributes and relationships. You can access the Sequence Number List Column for attributes through the applet menu option, Columns Displayed. The Sequence Number List Column is available for relationships in the standard Siebel application.

The sequence number for relationships determines the UI display order when no UI option is defined. In case of conflicting requirements between a desired evaluation order and the UI presentation order, always favor the evaluation order requirements. It is always possible to change the UI presentation order by using UI Option, UI Group, and then UI Item structure.

GoalMode Processing Order Example

The following hierarchy exists for CP 0 (CP = Customizable Product)

CP 0 GoalMode=1

Relationship01 Sequence Number: 1

CP 1

Relationship02 Sequence Number: 3

CP 2

Attr01 Sequence Number: 2

Attr02 Sequence Number: 4

CP 1 GoalMode=0 (or not set)

Relationship11 Sequence Number: 5

Product A

Relationship12 Sequence Number: 1

Product B

Attr11 Sequence Number: 15

Attr12 Sequence Number: 4

CP 2 GoalMode=2

Relationship21 Sequence Number: 5

Product C

Relationship22 Sequence Number: 1

Product D

Attr21 Sequence Number: 15

Attr22 Sequence Number: 4

Based on the GoalMode UI properties in the preceding hierarchy, Siebel Configurator resolves the solution in the following order:

CP0.Attr01

CP0.Attr02

CP0.Relationship01

CP 1.Relationship12

Product B

CP 1.Relationship11

Product A

CP1.Attr12

CP1.Attr11

CP 0.Relationship02

CP 2.Relationship22

Product D

CP 2.Attr22

CP 2.Port21

Product C

CP 2.Attr21

Using the GoalMode UI Property

The GoalMode UI property is especially useful for cascading formula calculations. For example, consider a formula with a constraint between 3 non-negative integer attributes: X, Y, and Z

Z >= (1000*X*(4*Y - 1))/(4*Y); for any X and any positive Y there is a set of valid values for Z,

Assume a user requests that X = 10.

If Siebel Configurator follows the order by which the variables are defined, Z is evaluated first. This might result in a long chain of trials and failures for the first couple of hundred values, and the calculation might exceed the allowed number of iterations.

If you explicitly specify that Siebel Configurator evaluates Y before evaluating Z, then the first choice value for Y results in finding a valid solution to the formula through propagation.

Another example might be a formula with two attributes X and Y with incompatible default values x and y, respectively. Consider the following statement that:

X = x excludes Y = y and conversely.

The default selections can not be satisfied for both attributes simultaneously and the result depends on the order. The GoalMode UI property enables you to specify explicitly the order in which the constrained variables are evaluated.

Setting Constraints for Numeric Attributes

In previous releases, you had to define a constraint if you wanted to constrain the range for a numeric (number and integer) attribute. There is now an easy, centralized way of restricting the upper and lower bounds for numeric attributes. You can specify the Minimum and Maximum range for an attribute through the column display in the Administration - Product screen, then the Attribute Definitions view or Product Classes view.

Minimum and Maximum values defined at the Attribute Definitions level restrict the domain of all Object (Classes and Products) Attributes, based on this definition.

Minimum and Maximum values defined at the Class Attribute level restrict the domain of all inherited Attribute of Objects derived from the Class.

Minimum and Maximum values can be overwritten for each level throughout the inheritance tree, with a restriction that derived objects can only reduce to the interval inherited from its parent, that is Minimum value is greater than or equal to the parent Minimum value and the Maximum value is less than or equal to the parent Maximum value.

Creating Groups of Related Configuration Constraints

Related constraints have the same basic construction but differ only in content. For example, you need to create a dozen exclude constraints of the form Product A excludes Product B. For each constraint the products will be different, but the basic construction is the same.

There are several processes you can use to create groups of related constraints. For example, you create twelve exclude constraints in the following ways:

- Create each constraint using the Exclude template in the “Pick a Constraint” tab of the Constraints view.
- Create the first constraint and save it. Then edit the constraint so that it becomes the second constraint and save the constraint. In the Save Rule form, click “Save changes as new Rule.” This creates the second of the twelve constraints. Repeat these steps to create the remaining constraints.
- Create the first constraint and save it. Then copy the constraint 11 times. Edit each of the copies. In the Save Constraint form, click Save to overwrite the copy with the changes.
- Create a constraint and save it as a template. Then select this template to create the remaining constraints.

Activating and Deactivating Configuration Constraints

When you create a constraint, it may be active or inactive by default:

- If you save the constraint using quick save the Active flag is checked automatically by default.
- If you save the constraint using the Save Constraint menu option, the user must select the Active check box, or else the rule is inactive.

An active constraint is used to compute all solutions (if the date is within the start and end date of the constraint).

When you deactivate a constraint, it is not used to compute solutions. One reason for deactivating a constraint is to help you test constraints in Validate mode. You can deactivate a group of constraints and then activate them one at a time to see how each affects the product's behavior when it is being configured.

Another reason to deactivate constraints, is when you want to release a version of a product that does not require a constraint to be active. You can deactivate the constraint and then release the product. The constraint is inactive in the released version and is not used to compute solutions.

To deactivate a constraint

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Constraints view.
4. In the Constraints list, select the desired constraint and deselect the Active checkbox.
5. From the Constraints List menu, select Validate.

   This starts a configuration session. Verify that the customizable product works correctly.

To activate an inactive constraint

1. In the Constraints list, select the desired constraint and select the Active checkbox.
2. From the Constraints List menu, select Validate.

   This starts a configuration session. Verify that the customizable product works correctly.

About Managing Configuration Constraints

You can manage constraints in the following ways:

- "Editing Configuration Constraints" on page 225
- "Copying Configuration Constraints" on page 225
- "Deleting Configuration Constraints" on page 226

Editing Configuration Constraints

In the Constraints List, you cannot edit the definition of the constraint in the Constraint column. To edit the definition, you must display the constraint in the Constraint Statement form, make your changes, and save the constraint. When you save the constraint, you can overwrite the constraint with the changes or save the changes as a new constraint.

The following procedure explains how to edit a constraint definition and overwrite the constraint with the changes.

To edit a constraint

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Constraints view.

   In the Constraints list, select the desired constraint.
4. From the Constraints list menu, select Edit Constraint.

   The Constraint Statement and “Insert a” tabs appear.
5. Edit and save the constraint.
6. From the Constraints List menu, select Validate.

   This starts a configuration session. Verify that the edited constraint works correctly.

Copying Configuration Constraints

When you copy a constraint, the application creates an exact duplicate of the constraint and displays it in the Constraints view. You can then edit the constraint definition as desired.

If you copy a constraint and make no changes to the copy, two exactly equivalent constraints are used to compute each solution. This does not cause a problem, and solutions are computed as if there was only one constraint.

Use the copy feature to create groups of constraints that are similar. Start by creating the basic constraint. Then copy it once for each constraint in the group. Finally, edit the copies to create the constraints in the group.

To copy a constraint

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Constraints view.
4. In the Constraints List, select the constraint you want to copy.

5 From the Constraints List menu, select Copy Record.

A copy of the constraint appears in the Constraints List. Its name begins with “Copy of.”

6 In the Name field, edit the constraint name as desired.

7 Edit the constraint statement, explanation, and start/end dates as desired.

8 From the Constraints List menu, select Validate.

This starts a configuration session. Verify that the new constraint works correctly.

Deleting Configuration Constraints

Deleting a constraint removes it from the customizable product. The template on which the constraint was based is not removed and remains available.

To delete a constraint

1 Navigate to the Administration - Product screen.

2 Select and lock the desired customizable product or product class.

3 Navigate to the Constraints view.

4 In the Constraints List, select the constraint you want to delete.

5 From the Constraints List menu, select Delete Record, and click OK when asked if you want to delete the record.

6 From the Constraints List menu, select Validate.

This starts a configuration session. Verify that the customizable product constraints function correctly.

Creating Configuration Constraint Templates

When you create and save a constraint, the constraint becomes part of the customizable product or product class. The constraint is not visible in other customizable products or product classes.

When you create and save a constraint as a template, it is added to the list of templates. The list of templates is visible in all customizable products or product classes. Create templates for those constraints that you will use with several customizable products. There is a single template list for customizable products and product classes, so any templates you create are available for both.

Templates that refer to items in one customizable product or product class cannot be used to refer to items in another customizable product or product class. Items include products, relationships, links, links and resources. For example, you write the following constraint for customizable product CP1 and save the constraint as a template called A Requires B:

Product A requires Product B

You also have customizable product CP2, that includes Product A and Product B. You want to write the same constraint for CP2.

If you use the template A Requires B in CP2, you will receive a validation error when you validate CP2. This is because each item in a customizable product receives a unique item ID. This item ID is what the application stores as the item name when you create a constraint or a constraint template in a customizable product. This ID is not transferable to other products with components.

NOTE: Constraint Templates cannot be edited or deleted. This is to prevent unintended problems across multiple products with components or product classes where templates have been used.

To create a constraint template

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Constraints view.
4. From the Constraints List menu, choose New Template.

   The Pick a constraint list appears.
5. Create the desired constraint that you want to use as a template.
6. Click Save Template and complete the necessary fields in the Custom Template Definition form, described in the following table.

Field	Comments
Name	Enter the template name. This name displays in the Pick a constraint list.
Template Identifier	Displays a text string that uniquely identifies the template. It does not display to users.
Description	Enter a description of what the template does. The description displays in the Pick a Constraint list.
Translation	Displays the internal translation of the template. This field is automatically populated by the application and can be used to construct the Spec.
Spec	Enter the constraint syntax. The constraint syntax can be derived from the Translation field.

Creating a Configuration Constraint Summary Report

You can obtain a report that lists all the configuration constraints in a customizable product or product class. The report shows the following information:

- Constraint name

- Constraint Statement
- Explanation
- Start date
- End date
- Active
- Updated date
- Updated by

The Constraint Summary displays in the Siebel Report Viewer. You can print the report or create an email attachment.

This report must be enabled on the report server before performing the following procedure.

TIP: The on-screen display of the report typically lists more products on each page than the Products list. Use the report to scan quickly through the product table.

To create a constraint summary report

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.

   If you omit this step, the most recently released version of the customizable product is loaded in the Constraints view.
3. Navigate to the Constraints view.
4. In the application icon bar, click the Reports icon.
5. In the Run Report applet, select Rule Summary.

15 Configuration Links

This chapter describes how to use the Links view to define and manage links. Links allow you to extract information from Siebel business components and from system variables and use it to write constraints.

For information about constraints, see Chapter 14, “Configuration Constraints.”

This chapter includes the following topics:

About Configuration Links

Links provide a way to use Siebel data in constraints that you write for a customizable product or product class.

For example, if you have clients outside the U.S., you could create a link that stores the account location. You could then write a constraint that uses the account location to determine what kind of power supply and plug types to include with a computer configuration.

The value of a link is determined when the user starts a configuration session and is not dynamically updated during the session.

Links can store the following types of information:

- Business component links store the value of a field in a Siebel business component.
- Context variable links store the value of a variable-map variable.
- System variable links store the value of a specific system variable.

Business Component Links

Business component links map a Siebel business component data field to a link name. The link name can then be used when writing constraints for a customizable product.

To create a business component link, you must have a thorough understanding of Siebel business components and be able to use Oracle’s Web Tools to identify business objects, business components, and field names.

When you define a business component link, the goal is to retrieve only one record. Several fields are provided in the link definition to help you do this. If more than one record is retrieved by the query, the link data is extracted from the first record in the group. If no records are retrieved by the query, the value entered in the default value field in the link definition is used.

You have the option to extract information from the current instance of a business component or from a new instance. For example, you select an account as part of creating a quote. You have defined a link for a complex product that extracts information from the business component that displays this record. When the user begins configuring the product, the link information will be extracted from the account record being used in the quote. The link uses the current instance of the business component.

Context Variable Links

Context variable links allow you to extract information from the current context by using the variable map. For example, if a variable in the variable map represents the Country field, you can use the context variable link to extract the name of the country in the current record.

To use context variable links, you must have an understanding of variable maps. For more information, see *Siebel Order Management Infrastructure Guide*.

In versions 7.8 and later, when Siebel Configurator is launched, the Context Service passes a property set to Siebel Configurator that is used for eligibility calculations and for pricing. The values in the property set are defined through Context Service administration, and they can be mapped so they are the same for quotes and orders.

Business component links are retained for backward compatibility, but context variable links are preferable because:

- They do not require an additional query to the database, improving performance.

They do not require you to define separate links for quotes and orders.

System Variable Links

Links can be defined to extract information from two system variables:

- **TODAY.** Returns today's date.
- **WHO.** returns the log-in name of the user who started the configuration session.

You can use the TODAY system variable to write time-sensitive constraints. For example, you create a link named TodayDate that stores the value of the TODAY system variable. You could then write a constraint that says if today's date is later than December 23, 2001, then the product 64 MB RAM is required in computer configurations.

You can use the WHO system variable to customize configuration constraints based on the user log-in name. For example, you create a link named UserName that stores the value of the WHO system variable. You could write a constraint that says if the user's log-in account name is jsmith, then 64 MB RAM is required in computer configurations.

Creating a Business Component Configuration Link

A business component link lets you extract information from Siebel business components and use it to write constraints.

To create a business component link you must know the business component name and field name containing the information you want to extract.

You must select and lock a customizable product before creating a link. When you create a link, it is automatically added to a picklist. You can then add the link to other customizable products by selecting it from the picklist.

When you create a link, it is added to a dialog box. You can copy this link definition to other customizable products and edit the link as needed. In turn, the edited link is added to the dialog box. When you remove a link from a customizable product, it is removed from the dialog box.

In some cases, you might also have to set the Link Specification and Force Active properties of the business component field that you are using in the link to TRUE.

When there are empty values for numeric or date attributes, Configurator constraint engine gives date fields the default value of today's date, and it gives numeric fields the default value of zero.

To create a business component link

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Linked Items view for the Work Space version.

4 In the Link Definitions list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter the name of the link. Use this name to refer to the link when you write constraints. This field is required.
BusObj Name	Enter the business object in which the business component resides. This field is required for business component links.
BusComp Name	Select the business component from which you want to extract information. This field is required for business component links.
BusComp Field Name	Select the field in the business component that contains the data you want to extract. This field is required for business component links.
Expression	Displays an XML expression that is automatically generated by your entries in the other fields.
Needs Execution	Select this check box to retrieve the link information from a new instance of the business component. Deselect it to retrieve information about the current record of the active business component in active business object before launching Siebel Configurator. **NOTE:** When this check box is deselected, the value of the link is always resolved with the active business component, irrespective of the calling mechanism. Deselecting this check box produces the same link value in Open UI, scripting APIs, and Web service interfaces.
Search Spec	Enter a Siebel query expression to narrow the search to one record. This field is evaluated only if you put a check mark in Needs Execution. An entry in this field is highly recommended.
Sort Spec	Enter a sort specification so that the desired record appears first if more than one record is retrieved. This field is evaluated only if you put a check mark in Needs Execution. An entry in this field is highly recommended.
Default Value	Enter the value that you want to assign to the link if the query returns no records. This field is highly recommended if you put a check mark in Needs Execution. If you are linking to a context variable, you must use this field to specify the default value of the context variable if the default value is not found in the context when the linked item is evaluated.
Keyword	For business component links, leave this field blank. This field is used for context variable links. For more information, see "Creating a Context Variable Link" on page 233.

Field	Comments
Context Variable	For business component links, leave this field blank. This field is used for context variable links. For more information, see "Creating a Context Variable Link" on page 233.
Context Variable Type	For business component links, leave this field blank. This field is used for context variable links. For more information, see "Creating a Context Variable Link" on page 233.
Description	Enter a description of what the link does. This description is not displayed to customers.

NOTE: To use an already-existing definition, click in the Name field and select the desired link definition from the dialog box.

Creating a Context Variable Link

A context variable link lets you extract information from the variable map "Cfg Eligibility Variable Map - Context" and use it to write constraints.

To create a context variable link you must use an existing variable definition from the variable map or add a new entry to the variable map. For more information, see "Context Variable Links" on page 230.

To create a context variable link

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Linked Items view for the Work Space version.
4. In the Link Definitions list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter the name of the link. Use this name to refer to the link when you write constraints. This field is required.
BusObj Name	For context variable links, leave this field blank. This field is used for business component links. For more information, see "Creating a Business Component Configuration Link" on page 231.
BusComp Name	For context variable links, leave this field blank. This field is used for business component links. For more information, see "Creating a Business Component Configuration Link" on page 231.
BusComp Field Name	For context variable links, leave this field blank. This field is used for business component links. For more information, see "Creating a Business Component Configuration Link" on page 231.

Field	Comments
Expression	Displays an XML expression that is automatically generated by your entries in the other fields.
Needs Execution	Select this check box to retrieve the link information from a new instance of the business component. Deselect it to retrieve information about the current instance of the business component.
Search Spec	For context variable links, leave this field blank. This field is used for business component links. For more information, see "Creating a Business Component Configuration Link" on page 231.
Sort Spec	For context variable links, leave this field blank. This field is used for business component links. For more information, see "Creating a Business Component Configuration Link" on page 231.
Default Value	Enter the value that you want to assign to the link if the evaluation returns no value. Please note that you can also specify a default value in the variable map definition. This default has precedence over the default defined here.
Keyword	To create a context variable link, choose Context Variable.
Context Variable	To create a context variable link, enter the name of the variable in this field. The variable must be in the variable map that is currently being used.
Context Variable Type	To create a context variable link, enter the variable type in this field.
Description	Enter a description of what the link does. This description is not displayed to customers.

Creating a System Variable Configuration Link

A system variable link lets you obtain the value of the following system variables and use it to write constraints:

- **TODAY.** Provides the system date. The data type is Date and can be used for date computations.
- **WHO.** Provides the user's login name. The data type is Text.

To create a system variable link

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Linked Items view for the Work Space version.

4 In the Link Definitions list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter the name of the link. Use this name to refer to the link when you write constraints. This field is required.
Keyword	Select either TODAY or WHO. This field is required for system variable links.
Description	Optionally, enter a description of what the link does. This description is not displayed to customers.

NOTE: Leave all the other fields blank. They are only used for business component links.

About Managing Configuration Links

You can manage links in the following ways:

- “Editing a Configuration Link Definition” on page 235
- “Deleting a Configuration Link” on page 235

Editing a Configuration Link Definition

You must select and lock a customizable product before editing a link definition. If you change the name of a link, the name is not changed in configuration constraints where it appears.

Editing the name of a link changes its name in the Pick Linked Item dialog box.

Deleting a Configuration Link

You delete a link for a customizable product by deleting the record from the Link Definitions list.

You must select and lock a customizable product before deleting a link.

16 Configuration Resources

This chapter explains how to create resources, which are variables that keep track of important configuration information when the user configures a customizable product. These variables can be used in configuration constraints.

For information about constraints, see Chapter 14, “Configuration Constraints.”

This chapter includes the following topics:

About Configuration Resources

Resources keep track of configuration variables that increase or decrease as the user configures a product. For example, suppose you are defining a desktop computer customizable product. The product includes several types of chassis. Each chassis has a different number of slots for expansion cards. Allowable configurations also include several types of expansion cards, such as disk controllers, and graphics cards.

You do not know in advance which chassis the customer will select or how many expansion cards. However, you do know that you must keep track of the number of slots during the configuration process to make sure that the customer configures the computer correctly.

Resources are the way you do this:

1. First define a resource to keep track of slots, for example slots-resource.
2. For the class containing all the chassis, define an attribute, slots-provided, that tells how many slots are in the chassis. Typically, this attribute will have a single-value domain and the data type will be Number.
3. For each class containing expansion cards, define an attribute, slots-required, that tells how many slots each card needs, usually 1. Typically, this attribute will have a single-value domain, and the data type will be Number.
4. Finally, write provide and consume constraint to manage the slots-resource.

When the user selects a chassis, a provide constraint adds the amount of the chassis’ slots-provided attribute to the slots-resource. When the user selects an expansion card, a consume constraint subtracts the amount of the card’s slots-required attribute from the slots-resource. In this fashion, the slots-resource keeps track of available slots in the computer chassis.

Resources definitions have the data type Number. This means that they can only have numeric, integer, or floating point values.

Creating Configuration Resources

When you create a resource, it is added to a dialog box. You can copy this resource definition to other customizable products and edit the definition as needed. In turn, the edited definition is added to the dialog box. When you remove a resource from a customizable product, it is removed from the dialog box.

All resources have the data type number. They can have only numeric, integer, or floating point values.

To create a resource

1. Navigate to the Administration - Product screen, then the Product Definitions or Product Classes view.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Resources view for the Work Space version.
4. In the Resource Administration list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Enter the resource name. Use this name to refer to the resource in configuration constraints. The resource name does not display to the user. Since a resource name can be used by more than one product, avoid making the name product-specific.
Description	Enter a description of the resource. This description does not display to the user.

Managing Resources Using Configuration Constraints

The most common way to manage a resource is to write provide and consume constraints that add or subtract the value of an attribute from the resource. For example, you could write a configuration constraint that contributes the number of slots in a chassis to a resource called slots available. You could also write configuration constraints that consume slots from the resource when the user picks an expansion card.

By convention, the value of a resource must exactly equal the sum of all the contributors to the resource. Constraints that consume or reduce the amount of a resource are negative contributors. The value of a resource is a computed value and cannot be directly set by a configuration constraint.

For example you define resource R. You then write a configuration constraint that sets the value of R to 5:

R ==5

When you validate the customizable product, this constraint will be rejected by the application because it sets the value of R at an arbitrary value rather than allowing the value of R to be computed as the sum of all its contributors.

About Managing Configuration Resources

You can manage resources in the following ways:

- “Editing Configurator Resource Definitions” on page 239
- “Deleting Configuration Resources” on page 239

Editing Configurator Resource Definitions

You must select and lock a customizable product before editing a resource definition. If you change the name of a resource, the name is not changed in configuration constraints where it appears.

Editing the name of a resource changes its name in the Pick Resource dialog box.

Deleting Configuration Resources

You delete a resource by deleting the resource record from the Resource Administration list. Deleting a resource from a customizable product deletes it from the Pick Resource dialog box.

You must select and lock a customizable product before deleting a resource.

17 Configuration Constraint Template Reference

This chapter explains the templates used when you create configuration constraints.

Before you read this chapter, you must understand the information about creating constraints in Chapter 14, “Configuration Constraints,” Chapter 15, “Configuration Links,” and Chapter 16, “Configuration Resources.”

This chapter includes the following topics:

About Configuration Constraint Processing

Unlike procedural languages like Siebel eScript or C++, constraints are elements of constraint programming. Constraint programming differs from procedural logic in several important ways.

In a procedural language, you write statements that are executed one after another. Control can be transferred to other parts of a program, but the method of execution remains serial. Procedural logic relies on groups of statements executed in order.

In constraint logic, you write constraints that describe what must be true about the solution. the Siebel Configurator engine organizes these constraints into a search plan and then searches systematically, trying different item values, until a solution that satisfies all the constraints is found. Constraints are evaluated in parallel rather than serial fashion. Their order of evaluation is unimportant.

For example, you write a series of configuration constraints that define all the ways a desktop computer can be configured. This is called the declarative portion of the customizable product, and the constraints are constraints on every solution. the Siebel Configurator engine requires that all constraints are observed in every solution. This is the key to understanding constraint programming: *all* constraints must be observed in *every* solution.

You then release this customizable product to users. Users interact with it by selecting components to configure a computer. Each item the user selects is treated by the Siebel Configurator engine as another constraint on the solution. These user-constraints are added to the constraints in the declarative portion of the product and are used to further narrow down the solutions the engine can create. If a user-constraint conflicts with a constraint in the declarative portion of the product, the user receives a message that provides options for resolving the conflict.

The Siebel Configurator engine must find a configuration that satisfies all the constraints in the declarative portion of the product, plus all those created by the user's choices (user-constraints). After the user adds or removes an item, the Siebel Configurator engine searches until it finds a solution that satisfies all the constraints, including the constraint created by the user's action. The Siebel Configurator engine then presents the solution. In many cases, the only thing that changes is that the item is added or removed.

However, other items may be added or removed depending on constraints in the declarative portion of the product. If the user selects an item and the Siebel Configurator engine cannot create a solution that satisfies all the constraints, the user is presented with an option to undo the current selection or previous selections so that all constraints can be satisfied.

Note that constraints are created both by the modeler and by the user. In the declarative portion of the product, the product administrator writes configuration constraints that define the relationships between items. For example, if item A is picked, item B is required. The user creates constraints by adding items. For example, picking item A creates a constraint that item A must be in the solution. The constraints that drive the solution are thus jointly provided by both the product administrator and the user.

It is important to understand that to produce a solution, the Siebel Configurator engine is free to do what is necessary to find a valid solution that satisfies all the constraints (declarative portion constraints and user-constraints). For example, the following are the only two constraints on items A and B in a customizable product:

The quantity of A < the quantity of B

The quantity of B != 4

If the user picks one A, the Siebel Configurator engine will require that there are at least two Bs in the solution. If the user then increases the quantity of A to two, the Siebel Configurator engine generates a solution in which there are at least three Bs. However, when the user adds the third A, instead of a solution that increases B by one, the new solution will have two more Bs, making the total number of Bs at least five. This occurs because both constraints must be satisfied, and because there is no constraint preventing the Siebel Configurator engine from generating solutions that increment B by more than one.

The following example illustrates that when there are several possible alternatives, the Siebel Configurator engine may choose any one of them. You have a customizable product with only the following two constraints on items A, B, and C:

Constrain A + B = C to be true
Constrain C = 1 to be true

In this example, either A or B can be zero, but not both. Neither can both be 1. the Siebel Configurator engine will choose either A or B and will actively exclude the other from the solution. For example, it could choose A and actively exclude B from the solution. It could also choose B and actively exclude A.

It is important when creating constraints to consider the domain of solutions the engine could produce. Otherwise, users may encounter unexpected results when they make selections. In the example above, you could write a constraint that prints a message to the user that A + B must equal 1 and then let users choose which option they want.

About Configuration Constraint Conditions

Many of the constraint templates contain conditions or expressions. For example:

- Exclude template: [Item or condition] excludes [item or condition]
- Require template: [Item or condition] requires [item or condition]

A condition is an explicit statement about the configuration. Conditions can play several roles in a constraint template. First, they can act as a test that determines whether a constraint is enforced. For example, you write the following constraint:

Item A > 4 excludes Item B

This constraint states that when the quantity of Item A is greater than 4 in the solution, then Item B cannot be present (is excluded). In this constraint, “Item A > 4” is a condition that, when true, causes Item B to be excluded.

When a condition is used as a test, the Siebel Configurator engine evaluates the condition and returns true or false. If the condition is true, the constraint is enforced.

Secondly, conditions can define a constraint. For example, you write the following constraint:

Item B excludes Item A > 4

This constraint states that when Item B is present in the solution, then the quantity of Item A in the solution cannot be greater than 4. In this constraint, "Item A > 4" is a condition that defines a constraint.

Conditions can take several forms:

- **Quantity comparisons.** The preceding examples are quantity comparisons.
- **Item values.** The value of attributes, linked items, and resources can be used as conditions.
- **Constraint Templates.** Constraint templates can be used as conditions. When constraint templates are used as conditions, the Siebel Configurator engine does not enforce the constraint as a constraint but instead evaluates the template as true or false. This is discussed further below.

Boolean operators (AND, OR, NOT) are provided in the Constraints List to allow combining conditions together or to negate an expression used in a condition.

A third way to use conditions is when writing require or exclude constraints about relationships. In the constraint "item A requires Relationship B" the Siebel Configurator engine has no way to determine which items in Relationship B to add to the solution if the user picks item A. So when the user picks item A, the Siebel Configurator engine prints a message in the user's message area stating that a selection from Class B is required.

Compound Logic and Comparison Operators in Configuration Constraints

Both Compound Logic and Comparison operators test for the truth of their operands. They return a true or false rather than a quantity.

Compound Logic operators, such as AND, NOT, OR, are used to link expressions together when creating a constraint. For example: (Condition A AND Condition B) requires Item C. Compound Logic operators are also called Boolean operators.

Table 16 presents the Compound Logic operators you can use with constraint templates.

Table 16. Compound Logic Operators

Operator	Example	Description
NOT	`! (A)`	Logical negation. True when *A* is false and false when *A* is true. *A* can be an item or sub-expression.
AND	`A AND B`	Both *A* and *B*. True only when both *A* and *B* are true. When used as a top-level constraint, means that only solutions where both *A* and *B* are true are allowed. *A* and *B* can be items or sub-expressions.
OR	`A OR B`	Either *A* or *B* or both. False only when both *A* and *B* are false. *A* and *B* can be items or sub-expressions.
Exclusive OR	`A XOR B`	*A* or *B* but not both. *A* and *B* must have opposite truth states. False when *A* and *B* are either both true or both false. *A* and *B* can be items or sub-expressions.
NOT AND	`! (A AND B)`	Converse of A AND B. False only when both A and B are true. When used as a top-level constraint, means that A and B cannot both be present. A and B can be items or sub-expressions.

Comparison operators compare their operands and return a true or false. In the following constraint, when the quantity of item A is less than item B (when the comparison is true), then item C is required.

(Item A < Item B) requires C

If you specify an item as an operand in a comparison, the quantity of the item in the solution is used to make the comparison. If you specify an expression as an operand, the expression must resolve to a number.

If you specify an expression that resolves to true or false, then true is assigned the value 1 and false is assigned the value 0.

Table 17 presents the Comparison operators you can use with constraint templates.

Table 17. Comparison Operators

Operator	Example	Description
Greater than	`A > B`	*A* is greater than *B*
Not less than	`A >= B`	*A* is greater than or equal to *B*
Equals	`A == B`	*A* equals *B*
Equals (compound)	`A==B==C==D`	True if A=B AND A=C And A=D
Not equal to	`A <> B`	*A* does not equal *B*

Table 17. Comparison Operators

Operator	Example	Description
Not greater than	A <= B	*A* is less than or equal to *B*
Less than	A < B	*A* is less than *B*

When you are building a constraint, you can compound the comparison operators. For example, you could build the following expression:

```
(A>B>C>D)
```

This expression is equivalent to the following expression:

```
A>B AND A>C AND A>D
```

Arithmetic Operators in Configuration Constraints

An arithmetic operator allows you to perform an arithmetic operation on two items. If an operand is a product, the value refers to the quantity of the product in the solution.

The operands in the expression can be two items or can be one item and a constant. For example, you can increase the quantity of an item by a constant amount.

If you specify an expression as one of the items, it must resolve to a quantity. If the expression resolves to true or false, then 1 is assigned to true, and 0 to false.

Results of calculations are handled differently for resources values than for product quantities. Calculation results for resources are expressed exactly, including a decimal point if necessary. Because product quantities represent discrete units, results involving them are rounded to the nearest integer.

Table 18 shows the arithmetic operators.

Table 18. Arithmetic Operators

Operator	Example	Description
Addition	A + B	Sum of *A* and *B*. *A* and *B* can be items or expressions. Result is floating point if A or B is floating point.
Subtraction	A - B	Subtracts *B* from *A*. *A* and *B* can be items or expressions. Result is floating point if *A* or *B* is floating point.
Negation	-(*A*)	Additive inverse of *A*. Uses only one operand. *A* can be an item or expression.
Multiplication	A * B	Product of *A* and *B*. Result is floating point if *A* or *B* is floating point. *A* and *B* can be items or expressions.

Table 18. Arithmetic Operators

Operator	Example	Description
Division	A / B	Quotient of *A* divided by *B*. Truncates ratio to integer if both *A* and *B* are integers. Result is floating point if *A* or *B* is floating point. *A* and *B* can be items or expressions.
Modulo	%(*A*, *B*)	Remainder of *A* divided by *B*. For example, %(1900, 72) results in 28. If *A* or *B* is floating point, the value is first rounded to the nearest integer; then the remainder is computed as for integers. *A* and *B* can be items or expressions.
Minimum	min(*A*, *B*)	Result is the smaller of *A* and *B* and is floating point if *A* or *B* is floating point. *A* and *B* can be items or expressions.
Maximum	max(*A*, *B*)	Result is the larger of *A* and *B* and is floating point if *A* or *B* is floating point. *A* and *B* can be items or expressions.

Table 19 shows additional arithmetic operators that also take numeric operands and produce numeric results. Use them to control numeric accuracy or change numeric characteristics.

Table 19. Additional Arithmetic Operators

Operator	Example	How Used
Integer	int(*A*)	Truncates *A* down to an integer. For example, if the operand is 6.7, returns 6. *A* can be an item or expression. Useful only with properties.
Float	flo(*A*)	Converts *A* to a floating point. It is the same as multiplying the operand by 1.0. *A* can be an expression.
Absolute value	abs(*A*)	Returns the absolute value of *A*. *A* can be an item or expression.
Sign test	sgn(*A*)	Returns -1 if the quantity of *A* <0, 0 if *A*=0, 1 if *A*>0. *A* can be an expression.

Attribute Value (Advanced) Template

The Attribute Value (Advanced) template has the following form:

(for [any] items)

You can toggle between [any] and [all]. The Attribute Value (Advanced) template can be used only within a require constraint, and it changes the logic for conditions involving attributes. This template does not display in the Pick a Constraint list. Instead, it displays in the list for inserting a condition.

The following require constraint contains two attribute conditions:

Attribute C = M in Relationship A requires Attribute D=P in Relationship B

This constraint works as follows: If *any* item from Relationship A has Attribute C=M in the solution, then *all* the items from Relationship B must have Attribute D=P in the solution. This is the default behavior of the require template when it contains attribute conditions and is called the Any-All form.

By inserting the Attribute (Advanced) template into the constraint, you can create all the other combinations of Any-All logic:

- Attribute C = M in Relationship A requires Attribute D=P (for any items) in Relationship B

  If any item from Relationship A has Attribute C=M in the solution, then there must be at least one item from Relationship B that has Attribute D=P in the solution (Any-Any form).

- Attribute C = M (for all items) in Relationship A requires Attribute D=P in Relationship B

  If all the items from Relationship A have Attribute C = M in the solution, then all the items from Relationship B must have Attribute D = P in the solution (All-All form).

- Attribute C = M (for all items) in Relationship A requires Attribute D=P (for any items) in Relationship B

  If all the items from Relationship A have Attribute C=M in the solution then there must be at least one item from Relationship B that has Attribute D=P in the solution (All-Any form).

NOTE: The *for all items* clause makes the condition evaluate to true when there are no items in the relationship. If you want the condition to evaluate to false even when there are no items in the relationship, use the advanced rule template and add an AND clause to the condition which checks that the relationship has at least one item in it.

To create this logic in other constraint types, such as exclude constraints, use one of the Advanced Constraint Templates to create the constraint. Then insert a numAttr condition in the constraint.

Conditional Value Template

The Conditional Value Template has the following form:

([value] when [condition] is true, otherwise [value])

This template allows you constrain a value based on a condition. The [value] can be any number or item in the customizable product.

This template does not display in the Pick a Constraint list. Instead, it displays in the lists for inserting arguments in a constraint. You can insert the Conditional Value template anywhere you can insert a number. The most common use for this template is with provide and consume constraints.

For example, you want product P1 to provide 2 to the resource R when the quantity of product P2 is greater than 10. If the quantity of P2 is not greater than 10, you want product P1 to provide 1 to resource R. You would write this constraint as follows:

P1 provides (2 when P2 >10, otherwise 1) to R

If you wanted product P1 to provide 2 to resource R only when product P2 is greater than 10, you would write this constraint as follows:

P1 provides (2 when P2 >10, otherwise 0) to R

Constrain Template

The Constrain template has the form:

Constrain [an expression] to be true

The constraint template is useful for making simple comparison or quantity expressions into top-level constraints. For example, you want make sure that there are exactly 4 of Item B in every solution:

Constrain [Item B = 4] to be true

You can also use the Constraint template to create exclude and require constraints that have only one operand. For example the following constraint excludes Item B from the solution:

Constrain [Item B =0] to be true

To require at least 1 Item B in the solution:

Constrain [Item B >= 1] to be true

By using Compound Logic operators, you can yoke conditions together and then allow or disallow the combination. For example, you want to make sure that if the quantity of Item A > 4 in the solution, then Item B must be less than 5, and conversely:

Constrain [Item A > 4 AND Item B < 5] to be true

Constrain Attribute Conditions Template

The Constrain Attribute Conditions template has the following form:

An attribute] [=] [a value] [requires or excludes] [an attribute] [=] [a value]

This template allows you to constrain the selectable values for one attribute based on the value the user selects for another attribute. The requires and excludes operators in this template work the same way as those in the Require and the Exclude templates.

Use this template when attribute choices for one item affect allowable attribute choices for another item. For example, you sell shirts in small, medium and large sizes. The user picks the size by selecting a value for the Size attribute. These shirts come in red, green, and blue for small and medium sizes. Large size shirts come in red only. The user picks the color by selecting a value for the Color attribute.

Use this template to write constraints that restrict the choices available to users when they pick a size or color. For example, you would write a constraint that both blue and green colors exclude the large size. If the user selects large for size, the blue and green attributes cannot be selected for color. If the user selects color blue, then large cannot be selected for size.

Constrain Attribute Value Template

The Constrain Attribute Value template has the form:

[An attribute] [=] [a value]

The Constrain Attribute Value template sets the value of an attribute so that it cannot be overridden by the user during a configuration session. If you write a constraint that sets the attribute equal to a value, this has the same effect as setting the attribute value and saving the record in Product Administration, Customizable Product, and then Product Attributes. By setting the comparison operator to other than equals (=), you can constrain the allowable ranges for numeric attribute values. For example you could write a constraint that constrains an attribute value to > 100. In this fashion you can use the Constrain Attribute Value template to validate user input for range-of-values attribute domains.

Depending on the data type of the attribute domain, the attribute value can be set to one of the enumerated types, to the value of a linked item, the value of another item's attribute, to a string, or to a number.

You can use this template to restrict attribute values based on conditions that occur during a configuration session. For example, you could write a constraint that restricts one attribute's value if the user chooses a specified value for another attribute.

This template cannot be used to constrain the attributes of customizable products that are components in a customizable product. For example, customizable product CP1 has as one of its components customizable product CP2. You cannot use this template to constrain the values of attributes in CP2.

If the product administrator has set the value of an attribute in the Product Attributes list, this value cannot be overridden by a configuration constraint, or by the Siebel Configurator engine.

Constrain Conditionally Template

The Constrain Conditionally template has the form:

> When [condition A is true] [enforce constraint B], otherwise
> [enforce constraint C]

This template provides if-then-else logic. When the condition is true the first expression is enforced as a constraint. If the condition is false, the second expression is enforced as a constraint.

Another way to view the logic is as a relationship between a condition and two constraints:

- If condition A is true, then B is enforced as a constraint, and C can be either true or false.
- If condition A is false, then C is enforced as a constraint, and B can be either true or false.

The condition can be defined as a quantity comparison of a product, relationship, or resource. It can also be the value of a linked item. Compound logic operators (AND, OR, and so on) are provided to link conditions together.

The constraints can also be quantity comparisons of products, relationships, or resources. The value of a linked item can also be used.

Constrain Product Quantity Template

The Constrain Product Quantity template has the form:

[The quantity of a product] [=] [a value]

The template has three parts. The first operand names the product.

The operator, [=], defines how the product quantity is related to the third operand, [a value]. The operator is limited to numeric comparisons (=, <, >, and so on).

The [a value] operand can be any of the following:

- The quantity of a product in the solution
- The quantity of items from another relationship in the solution
- The quantity of items in the solution from a class within a relationship
- The value of an attribute (the data type for the attribute must be number or integer)
- A number

The “Insert a” tab provides two sets of arithmetic functions that allow you to combine these items. For example, you could write the following constraint:

The quantity of Item A from Items Class <= the quantity of Products-Class items

In this constraint, “the quantity of item A from Items Class” is the first operand. It names Item A in the Items class. The second operand is “<=”.

The third operand is “the quantity of Products-Class items” and refers to items in the Products class. This class is contained within a relationship in the customizable product.

The constraint states that the quantity of Item A in the solution must be less than or equal to the number of items from the Products-Class in the solution.

Use Constrain Product Quantity constraints to set limits on the quantity of products that can be in the solution. The limits can be defined as a number or as the quantity of other items, or the value of an attribute.

Constrain Relationship Quantity Template

The Constrain Relationship Quantity template has the form:

[The quantity of a relationship] [=] [a value]

The quantity of a relationship is the total number of items that have been added to the solution from the relationship. For example, a relationship contains Item A and Item B. If there is one Item A and one Item B in the solution, then the relationship quantity is two. If there are two of Item A in the solution and no Item Bs, the relationship quantity is also two.

The operator, [=], defines how the relationship quantity is related to the third operand, [a value]. The operator is limited to numeric comparisons (=, <, >, and so on).

The [a value] operand can be any of the following:

- The quantity of a product in the solution
- The quantity of items from another relationship in the solution
- The quantity of items in the solution from a class within a relationship
- The value of an attribute (the data type for the attribute must be number or integer)
- A number

The “Insert a” tab provides two sets of arithmetic functions that allow you to combine these items into expressions. For example, you could write the following constraint:

The quantity of Items from Items relationship <= (2 * [Training Classes “Intro Course” Items-limit-attribute])

The first operand is the relationship: “The quantity of Items from Items relationship.” The operator is “<=”. The second operand is the “Items-limit-attribute.”

This constraint states that the quantity of items from the Items relationship must be less than or equal to twice the value of the “Items-limit-attribute” of the Intro Course. The Intro Course is located in the Training Classes relationship.

Use Constrain Relationship Quantity constraints to set limits on the quantity of products from a relationship that can be in the solution. The limits can be defined as a number or as the quantity of other items, or the value of an attribute.

Constrain Resource Value Template

The Constrain Resource Value template has the form:

[A resource] [=] [a value]

The template has three parts. The first operand names the resource.

The operator, [>], defines how the resource is related to the second operand, [a value]. The operator is limited to numeric comparisons (=, <, >, and so on).

The [a value] operand can be any of the following:

- The quantity of a product in the solution
- The quantity of items from another relationship in the solution
- The quantity of items in the solution from a class within a relationship
- The value of an attribute (the data type for the attribute must be number or integer)
- A number

The “Insert a” tab provides two sets of arithmetic functions that allow you to combine these items into expressions.

Use the Constrain Resource Value template to define limits on the value of a resource. For example, you can constrain a resource to be greater than 0 and less than 5.

Display Message Template

The Display Message template has the following form:

[Item or condition] displays this rule's explanation

The Display Message template is one of two ways to communicate messages to the user. The other is the Display Recommendation template.

In the Display Message template, when the condition is true, the explanation you entered for the constraint is displayed to the user in the Message area. The explanation must describe the condition in the constraint and can add additional information.

Use the Display Message template to display messages when a defined condition is true.

For example, you write the following Display Message constraint:

When [Product A is selected], display this rule's explanation.

You then save the constraint and enter the following in the Explanation field:

Product A has been selected. You can purchase only two of these items.

During a configuration session, when the user selects Product A, the constraint explanation appears in the message area.

Display Recommendation Template

The Display Recommendation template has the following form:

[Item or condition] recommends [item or condition] by displaying this rule's explanation.

The Display Recommendation template displays the constraint's explanation based on the truth state of its two item/condition operands, as shown in Table 20.

Table 20. Truth Table for Display Recommendation Template

First Item / Condition	Second Item / Condition	Message Displays?
True	True	No
False	True	No
True	False	Yes
False	False	No

The table shows that the constraint explanation displays only when the first item/condition is true and the second is false. The recommendation displays in the user's message area.

The Boolean equivalent of how the Display Recommendation template works is shown below. The expression is true only when A is true and B is false. When the expression is true, the message displays.

NOT ((NOT A) OR B)

Use Display Recommendation constraints to up-sell or cross-sell other configuration options, to inform the user of configuration restrictions, or to offer configuration suggestions.

For example, when the user picks Product A, you want to display a message recommending Product B. If the user then picks Product B, you want to stop displaying the message.

You would write this constraint as follows:

Product A recommends Product B by displaying this rule's explanation.

In the constraint's Explanation field: When you select Product A, we recommend you also purchase Product B.

Exclude Template

The Exclude template has the form:

[item or condition] excludes [item or condition]

The exclude constraint is mutual. For example, if Item A is present in the solution, Item B cannot be present. Conversely, if Item B is present, Item A cannot be present.

The excludes constraint is functionally equivalent to a Boolean ! (A AND B), that is, a NAND operator. In Table 21, a T (true) means the item is present in the solution. An F (false) means it is not present or is excluded.

Table 21. Truth Table for Exclude

A	B	A AND B	! (A AND B)
T	F	False	True
F	T	False	True
F	F	False	True
T	T	True	False

The truth table shows that an exclude constraint is always true except when both operands are present in the solution.

Use an exclude constraint to:

- **Prevent technical configuration errors.** For example, a computer operating system or software application may be incompatible with certain microprocessors.

- **Prevent configurations that are undesirable or ineffective.** For example, in a financial model, an exclude constraint could prevent adding a low quality bond fund to a retirement portfolio.

Items

An item in an exclude constraint can be any of the following:

- A relationship or class within a relationship
- A product within a relationship or class

Here is how the exclude constraint works with items:

- **Product A excludes Product B**

  If Product A is present in the solution, then Product B cannot be present. If Product B is present in the solution, then Product A cannot be present.

- **Relationship A excludes Product B**

  If any product in Relationship A is present in the solution, then Product B is excluded. If Product B is present, then no product from Relationship A can be present.

- **Relationship A excludes Relationship B**

  If any Product in Relationship A is present in the solution, then no product in Relationship B can be present. If any product in Relationship B is present in the solution, then no product from Relationship A can be present.

Conditions

Conditions in an exclude constraint can take many forms. For example, an attribute condition specifies an attribute value. Here is a summary of how the exclude constraint works with conditions in general. How exclude constraints work with specific types of conditions is covered in the topics following this one.

- **Product A excludes Condition B**

  If Product A is present in the solution, then Condition B cannot be true. If Condition B is true in the solution, then product A cannot be present.

- **Relationship A excludes Condition B**

  If any product in Relationship A is present, Condition B cannot be true. If Condition B is true in the solution, then no product from Relationship A can be present.

- **Condition A excludes Product B**

  If Condition A is true in the solution, then Product B is excluded. If Product B is present in the solution, then Condition A cannot be true in the solution.

- **Condition A excludes Relationship B**

  If Condition A is true in the solution, then all products in relationship B are excluded. If any product from Relationship B is present in the solution, then Condition A cannot be true in the solution.

- **Condition A excludes Condition B**

  If Condition A is true in the solution, then Condition B cannot be true in the solution. If Condition B is true in the solution, then Condition A cannot be true in the solution.

Attribute Conditions

Attribute conditions are used to exclude specific attribute values for an item or group of items. For example, if the user picks item A, then the "Large" attribute value for item B is excluded.

In the constraint examples below, the attributes are defined on items in relationships within the customizable product. You can also define exclude constraints on the attributes of the customizable product itself.

In the following constraint examples, *excluded* means the user can no longer select the item. If the excluded item is a relationship, the user can no longer select any of the products in the relationship. Excluded also means the Siebel Configurator engine will not create solutions that contain the excluded item.

- **Product A excludes Attribute C = M in Relationship B**

  If Product A is present in the solution, then the value M will not be selectable for Attribute C in Relationship B.

  If any product with Attribute C = M in Relationship B is present in the solution, then Product A is excluded from the solution.

- **Relationship A excludes Attribute C = M in Relationship B**

  If any product in Relationship A is present in the solution, then the value M will not be selectable for Attribute C in Relationship B.

  If any product with Attribute C = M in Relationship B is present in the solution, then all the products in Relationship A are excluded from the solution.

- **Attribute C = M in Relationship A excludes Product B**

  If any product with Attribute C = M in Relationship A is present in the solution, then Product B is excluded from the solution.

  If Product B is present in the solution, then the value M will not be selectable for Attribute C in Relationship B.

- **Attribute C = M in Relationship A excludes Relationship B**

  If any product with Attribute C = M in Relationship A is present in the solution, then all the products in Relationship B are excluded from the solution.

  If any products from Relationship B are present in the solution, then the value M will not be selectable for Attribute C in Relationship B.

- **Attribute C = M in Relationship A excludes Attribute D = P in Relationship B**

  If any of the products with Attribute C = M in Relationship A are present in the solution, then the value P for Attribute D will not be selectable in Relationship B.

  If any products with Attribute D=P in Relationship B are present in the solution, then the value M will not be selectable for Attribute C in Relationship A.

Quantity Conditions

Quantity conditions compare the quantities of two items. Depending on the session context in which a quantity condition is evaluated, it either returns true/false or is enforced as a constraint.

For example you write the following configuration constraint,

(Product A > Product B) excludes Product C

Context A. If the user picks Product A so that its quantity is greater than Product B, then Product C is excluded. In this case, the quantity condition is evaluated as true/false, and Product C is excluded when it is true.

Context B. If the quantity of Product A in the solution is not greater than Product B and the user picks Product C, then the quantity condition is enforced as a constraint. In all further solutions, the Siebel Configurator engine will require that the quantity of Product A is < = Product B if Product C is present.

Other Item Constraints

Several other types of conditions can also be used in exclude rules. In some cases these conditions do not make sense when used as the second operand in an exclude rule. Table 22 summarizes how to use item constraints.

Table 22. Item Constraint Usage in Exclude Rules

Item Constraint	First Operand	Second Operand
Attribute Value	Yes	Yes
Consumes	No	No
Provides	No	No
Excludes	Yes	Yes
Excludes List	Yes	Yes
Requires	Yes	Yes
Requires List	Yes	Yes
Linked Item Value condition	Yes	Yes
Message, Recommends	No	No

Table 22. Item Constraint Usage in Exclude Rules

Item Constraint	First Operand	Second Operand
Product Quantity	Yes	Yes
Relationship Quantity	Yes	Yes
Class Quantity	Yes	Yes

Nested Expressions as Conditions

The Exclude Template can itself be used as a condition in other constraints. The most common templates used for writing nested constraints are the Exclude and Require templates. For example you could write the configuration constraint:

Product A excludes (Product B excludes Product C)

The Boolean form of this constraint is as follows:

A NAND (B NAND C)

In Table 23, a T (true) means the item is present in the solution. An F (false) means the item is not present or is excluded.

Table 23. A AND ! (B AND ! (C))

Row	A	B	C	(B AND ! (C))	A AND ! (B AND ! (C))
1	F	F	F	T	T
2	F	F	T	T	T
3	F	T	F	T	T
4	F	T	T	F	T
5	T	F	F	T	F
6	T	F	T	T	F
7	T	T	F	T	F
8	T	T	T	F	T

The truth table lets you analyze what happens when the user picks items. For example, there are no Product A, Product B, or Product C in the solution. The user picks Product B. You evaluate how the Siebel Configurator engine will respond as follows:

1. Picking B means that B is true, so eliminate all rows from the table where B is False. This leaves rows 3, 4, 7, and 8.
2. The engine returns solutions in which all constraints are true, so you can eliminate any of the remaining rows where the whole constraint is false. This means you can eliminate row 7. This leaves rows 3, 4, and 8.

3 Now examine the truth conditions for Product A and Product C in the rows 3. 4, and 8. The table shows that A is false in row 3 and 4 but true in row 8. This means that if the user picks B, then A can be either present or absent. It is not constrained. The table shows that C is false in row 3 but true in rows 4, and 8. This also means that C is not constrained. Thus, the user can pick B without A or C being excluded or required.

If A or C had been true in all three remaining rows, this means A or C is required. If A or C had been false in all three rows, this means A or C is excluded.

Multiple Operands

You can add multiple operands to an exclude constraint by clicking the Compound Field button when you create the constraint.

For example, you could create the constraint:

Item A excludes Item B > 2, Item C < 5

This constraint means the following:

- If Item A is present in the solution, Item B cannot be greater than 2
- If item A is present in the solution, Item C cannot be less than 5
- If Item B is greater than 2 in the solution, Item A cannot be present
- If item C is less than 5 in the solution, Item A cannot be present

Using commas to separate expressions is the same as writing two constraints:

Item A excludes Item B > 2

Item A excludes Item C < 5

If you want to write a constraint where you exclude the *combination* of two conditions you would do it as follows:

Item A excludes (Item B > 2 AND Item C < 5)

This constraint means that if Item A is present in the solution, the two conditions cannot be simultaneously true in the same solution. If Item A is present, the quantity of Item B can be greater than 2 as long as the quantity of Item C is not less than 5 and conversely.

Procedural Condition Templates

You can use two procedural condition rule templates together:

- “Procedural-Condition Check Template” on page 260
- “Set Procedural Conditional Variable Template” on page 261

For information on how to use these templates, see “Example of Using Procedural Condition Rules” on page 261.

Procedural-Condition Check Template

The Procedural-Condition Check template has the form:

If procedural condition [item or condition] is true, enforce resultant condition item or condition

You use the Procedural-Condition Check template to check for a precondition. If the item or condition it checks for is true, then the resultant condition is enforced in the solution.

If condition 1 is satisfied before the user clicks, condition2 must be satisfied after the user clicks. Otherwise, an undo only conflict error message is generated. During LoadInstance, this rule is not enforced, because no precondition is satisfied.

For example, consider the following scenario:

Root Product

|_Relationship 1

|_Product A

|_Product B

|_Product C

Constraint: If the procedural condition saying that the quantity of Product A from Relationship 1 equals 1 is true, then enforce the resultant condition saying that the quantity of Product B from Relationship1 equals 1.

Expected Behavior: When Product A is selected and has a quantity of 1, user must select Product B and enter the quantity of 1 before performing any other operation. If the user makes a selection other than Product B, a conflict message is displayed.

The procedural condition applies only to relationships and attributes that are referenced by constraints. For example, consider the following scenario:

Root Product

|_Relationship 1

|_Product A

|_Product B

|_Product C

|_Relationship 2

|_ Product D

|_ Product E

Constraint: If procedural condition the quantity of Product A from Relationship 1 = 1 is true, enforce resultant condition the quantity of Product B from Relationship 1 = 1.

When the user selects Product A and then selects Product C, a conflict message is displayed, because Relationship 1 is referenced within the constraint and relationship items are loaded by the constraint engine. When the user selects Product A and then selects Product D, a conflict message is not displayed, because Relationship 2 is not referenced by any constraint.

Set Procedural Conditional Variable Template

The Set Procedural Conditional Variable template has the form:

[Item or condition] sets Procedural condition variable value to true

This template defines a variable on the product instance. This variable's name is the rule name, and its value is evaluated after each user click. The variable can be retrieved from the Siebel Configurator service using a script and can be used to control the application behavior dynamically.

Excluding Values in the Administration - Product Screen

You can specify in the Administration - Product screen that certain values are excluded based on the expression variables defined by Set Procedural Conditional Variable rules. You can do this using the property, CondExcl=exprVarName. The attribute values or products in the domain with this property are shown as excluded when exprVarName evaluates to true.

To exclude values based on Set Procedural Condition Variable rules

1. Navigate to Administration - Product screen, then the Products Definitions view.
2. In the Versions list, drill down on the Workspace version of the product.
3. Click the Properties view tab.
4. In the Properties list, add a new record and complete the necessary fields, as described in the following table.

Field	Comments
Name	Enter CondExcl.
Value	Enter an expression variable. The attribute values or products in the domain with this property are shown as excluded when this variable evaluates to true.

Example of Using Procedural Condition Rules

For example, consider the following scenario:

- Customizable product A has a relationship R1 with products B1 and B2.
- Product A has attribute C with enumerated values C1 and C2. The option C2 is valid only with product B2.
- You want to show the C2 option even with the B1 product choice, and then display an appropriate message if C2 is chosen.

Set up the following constraint:

```
If procedural condition NOT (selection of [B2]) is true, then enforce resultant
condition NOT (the attribute C = C2)
```

Provide and Consume Templates

The Provide template has the form:

[An item] provides [a value] to [a target]

The Consume template has the form:

[An item] consumes [a value] from [a target]

Provide and consume constraints positively or negatively increment the amount of the target operand each time the specified item is added to the solution. Provide constraints contribute a positive amount; that is, they increase the amount of the target. Consume constraints contribute a negative amount; that is, they reduce the amount of the target.

Contrast this behavior with the behavior of the require constraints. For example, Item A requires Item B. The first time the user picks Item A, if no item B is in the solution, the Siebel Configurator engine adds at least one Item B. The second time the user picks Item A, the engine does not increment Item B because the require constraint does not consider the quantity of Item A in the solution, only that Item A is present.

Now consider the constraint Item A provides 1 to Item B. Each time the user picks Item A, the Siebel Configurator engine increments the number of Item B in the solution by 1. This constraint ties the quantities of Item A and Item B together so that each Item A requires an Item B. Provide and consume constraints work directly with quantities expressed as resource or attribute values, while require constraints consider only the presence or absence of an item.

NOTE: Provide and Consume constraints work properly when the resource is defined in the parent product. For modeling scenarios that require embedding resources in the child product, use attributes of type Integer instead.

Item

The Item operand can be a product, a relationship, a product class within a relationship, a resource or an attribute. If the item is a relationship or class, the constraint applies to all the items in the relationship or class. For example, Relationship A provides 1 to Item B. Each time an item from Relationship A is added to the solution, Item B is added to the solution.

Value

The value operand defines the quantity to be contributed to the target. The Constraints view provides several methods for determining this quantity:

- You can explicitly state the quantity, for example Item A consumes 1 from Item B. This constraint means that each Item A added to the solution decreases the amount of Item B in the solution by 1.
- You can define the value as the quantity of another item, or the value of an attribute, linked item or resource. For example, Item A provides three times the quantity of Item B to Item C. This constraint means that for each Item A added to the solution, the quantity of Item C is incremented by three times the quantity of Item B.

- You can define an expression that determines which products in the relationship or class specified in Item will increment the target. For example: Any item of Relationship A provides Relationship A, Attribute Color = Red to Item B. This constraint means that for each item in Relationship A for which the attribute Color = Red that is added to the solution, the quantity of Item B is incremented by one.

Target

The target operand is incremented by the amount specified in the value operand. The target can be a product, resource, or product attribute. It cannot be a relationship, a class, or an expression.

Product Target

When the target operand is a product, the quantity of the product is incremented. For example, Product A provides 2 to Product B. This constraint means that each Product A added to the solution increases the quantity of Product B (the target) by 2.

The consume constraint works the same way. For example, Product A consumes 2 from Product B. This constraint means that each Product A added to the solution decreases the quantity of Product B (the target) by 2.

Resource Target

When the target is a resource, the value of the resource is incremented. One of the most common uses of provides and consume constraints is to manage resources.

Resources keep track of configuration variables that increase or decrease as the user makes selections. For example, suppose you are creating a customizable product rule for configuring desktop computers. Your product includes several types of chassis. Each chassis has a different number of slots for expansion cards. The product also includes several types of expansion cards, such as disk controllers, and graphics cards.

You do not know in advance which chassis the customer will select or how many expansion cards. However, you do know you must keep track of the number of slots available in a chassis during the configuration process to verify that the computer is configured correctly.

Using provide and consume constraints to increment a resource is the way to handle this:

- First define a resource to keep track of slots, for example Slots Available.
- Then define an attribute called Slots on the chassis class. Create a enumerated values of data type integer. Create one record for each chassis type. The value for each record is an integer equal to the number of slots in the chassis type. This creates a menu of choices for setting the number of slots in a chassis. Assign the list of values to the Slots attribute definition.
- Display the attributes for each chassis and set the value of the Slots attribute and save the record. This sets the number of slots in the chassis and prevents it from being changed by the user or the Siebel Configurator engine.
- Add the chassis class to a relationship, Chassis.

- Define an attribute called Slots Required on the expansion card class. Use a range of values domain and set the data type to Integer. Enter: ==1 as the validation expression. Enter 1 as the default value.
- Display the attributes for each expansion card and save the Slots Required record. This sets the number of slots required at 1 and prevents it from being changed by the user or the Siebel Configurator engine.
- Add the expansion card class to a relationship, Expansion Card.
- Write the following constraint: Chassis provides Chassis Slots to Slots Available.
- Write the following constraint: Expansion Card consumes Expansion Card Attribute = Slots Required from Slots Available.
- Write constraints as needed to determine what happens if Slots Available is 0 or if it becomes negative.

When the user selects a chassis, the provide constraint increases the Slots Available resource by the number of slots in the chassis. Each time the user selects an expansion card, the consume constraint decreases the Slots Available resource by one. Thus, the Slots Available resource maintains a record of how many slots are available in the chassis during the configuration session.

If a resource has the same name in two different customizable products, the Siebel Configurator engines treats them as the same resource. You can take advantage of this in cases where one product with components is contained within another. For example, product with components CP2 is contained within product with components CP1. You define resource R1 in both products. Constraints in either product with components that contribute to R1, affect the value of R1 in both products. Use this behavior to allow a parent product with components to contribute to a resource in a child product with components.

Attribute Target

When the target is an attribute, the value of the attribute is incremented. Attribute targets are very similar in behavior and use as resource targets. There are several restrictions on using provide and consume constraints to manipulate attribute values:

- The data type of the attribute must be numeric (Integer or Number).
- The attribute must be available for manipulation. You must not have set the value of the attribute. You do this by selecting an attribute value and saving it in Product administration, Customizable Products, and then Product Attributes.
- A child product with components can contribute to attributes defined on the parent. The parent cannot contribute to attributes defined on the child product. For example, product with components CP2 is contained within product with components CP1. CP2 can contribute to attributes defined within CP1. CP1 cannot contribute to attributes defined within CP2.

Use attributes as targets instead of defining multiple resources that keep track of similar variables. This ties the variables directly to a class and makes it easier to keep track of the variables' roles.

Simple Provide and Consume Templates

The Simple Provide template has the form:

Provides [a value] to [a target]

The Simple Consume template has the form:

Consume [a value] from [a target]

The Simple Provide and Simple Consume templates positively or negatively increment the amount of the target operand. These templates are intended for use as the action portion of a conditional constraint. If the condition is true, then a value is provided or consumed from the specified target.

The Simple Provide template contributes a positive amount, that is it increases the amount of the target. The Simple Consume template contributes a negative amount, that is it reduces the amount of the target.

Value

The value operand defines the quantity to be contributed to the target. The Constraints view provides several methods for determining this quantity:

- You can explicitly state a number as the value.
- You can define the value as the quantity of another item, or the value of an attribute, linked item or resource. For example, you can provide three times the quantity of Item B to Item C.
- The value can be an expression that resolves to an amount. This amount is then contributed to the target. For example: When a condition is true, it provides Relationship A, Attribute Color = Red to Item B. This constraint means that when the condition is true, then each item in Relationship A for which the attribute
  Color = Red that is added to the solution, increments the quantity of Item B by one.

Target

The target operand is incremented by the amount specified in the value operand. The target can be a product, resource, or product attribute. It cannot be a relationship, a class, or an expression.

Product Target

When the target operand is a product, the quantity of the product is incremented in the solution.

Resource Target

When the target is a resource, the value of the resource is incremented.

Attribute Target

When the target is an attribute, the value of the attribute is incremented. Attribute targets are similar in behavior and use as resource targets. There are several restrictions on using provide and consume constraints to manipulate attribute values:

- The data type of the attribute must be numeric (Integer or Number).
- The attribute must be available for manipulation. You must not have set the value of the attribute. You set the value of an attribute by selecting an attribute value and saving it in Product Administration, Customizable Products, and then Product Attributes.
- A child product with components can contribute to attributes defined on the parent. The parent cannot contribute to attributes defined on the child product. For example, product with components CP2 is contained within product with components CP1. CP2 can contribute to attributes defined within CP1. CP1 cannot contribute to attributes defined within CP2.

Relationship Item Constraint Template

The Relationship Item Constraint template has the form:

For each item [in a relationship], constrain [an expression] to be true

The “in a relationship” operand can be a whole relationship, a subclass of items in a relationship, or a product in a relationship. The “an expression” operand can be any constraint template or any constraint you construct from templates.

The purpose of the Relationship Item Constraint template is to allow you to write a constraint for items in a relationship as if you had written the constraint separately for each instance of the items. For example, you define Relationship A that contains the product with components desktop PC. The desktop PC is a product with components that includes two relationships: CPU and Hard Drive. You then write the following constraint:

For each item in Relationship A, constrain CPU requires Hard Drive to be true

This constraint enforces “CPU requires Hard Drive” separately on each instance of desktop PC in Relationship A. All the desktop PCs from Relationship A must have a hard drive if they have a CPU.

A require constraint does not do this. Suppose you had written the following constraint:

CPU requires Hard Drive

This constraint means if any desktop PC has a CPU from the CPU relationship then at least one desktop PC must have a hard drive from the Hard Drive relationship.

This means, for example, that if the user configures three desktop PCs, all with CPUs, then only one of them must have a hard drive. If the user removes the hard drive, the Siebel Configurator engine would add a hard drive to another desktop PC in the solution or add a new desktop PC that contains only a hard drive. The require constraint defines a constraint that is true about the group of desktop PCs in the solution rather than about individual desktop PCs.

Another problem with the require constraint is that it does not limit enforcement of the constraint to the items in Relationship A. If, in the require constraint example, desktop PCs were also contained in Relationship B, then desktop PCs configured from Relationship B would also be considered when enforcing the require constraint for desktop PCs configured from Relationship A.

An important use of this template is to write constraints that apply to products with components only when these products are contained in a relationship within another product with components.

Require Template

The Require template has the form:

[item or condition] requires [item or condition]

A require constraint is a logical implies. If the first operand is true then the second operand is implied (must be true). For example, Item A requires Item B. This constraint means that if Item A is present in the solution, then Item B must be present. Another example: Condition A requires Item B. This constraint means that if Condition A is true, then Item B must be present in the solution.

A require constraint is not mutual. The constraint Item A requires Item B does not imply Item B requires Item A. However, if Item B is excluded then Item A is also excluded. This is because when Item B is excluded, the constraint can never be true.

The require operator is functionally equivalent to the following Boolean expression:

(NOT A) OR B

In the first two columns of Table 24, a 1 means the item is present in the solution. A 0 means it is absent or excluded.

Table 24. Truth Table for Require Expressions

A	B	NOT A	(NOT A) OR B
1	0	0	False
0	1	1	True
0	0	1	True
1	1	0	True

The table shows that a require constraint behaves as follows:

- False when Item A is present and Item B is not. (If Item B cannot be present, Item A cannot be present).
- True when Item B is present and Item A is not.
- True when neither is present.
- True when both are present.

Use require constraints to:

- To create a requires relationship for items in different relationships. For example, you can write a constraint that if Item A in relationship 1 is picked, then Item B in relationship 2 is required.
- To add items to the configuration if a condition is true.
- To create relationships between other items (conditions) when a product is added to the solution.

Items

An item can be any of the following:

- A relationship or class within a relationship. For example, Relationship A requires Item B. This constraint means that when any product in relationship A is present in the solution, then Item B must be present.
- A product within a relationship

When items are the operands, require constraints are concerned only with presence or absence, not quantity. For example, Item A requires Item B. When the first Item A is added to the solution, the Siebel Configurator engine will add at least one Item B if none are present. When the second Item A is added, the engine does not add any more Item B, since at least one is already present.

Here is how the require constraint works with items:

- **Product A requires Product B**

  If Product A is present in the solution, then Product B is required. If Product B is excluded, then Product A is excluded.

- **Relationship A requires Product B**

  If any product in Relationship A is present in the solution, then Product B is required. If Product B is excluded, then all the products in Relationship A are excluded.

- **Relationship A requires Relationship B**

  If any Product in Relationship A is present in the solution, then at least one product in Relationship B must be present. If all the products in Relationship B are excluded, then all the products in Relationship A are excluded.

Conditions

Conditions can take many forms. Here is a summary of how the require constraint works with conditions in general. How require constraints work with specific types of conditions is covered in the topics following this one.

- **Product A requires Condition B**

  If Product A is present in the solution, then Condition B is required to be true. If Condition B is false, then Product A is excluded.

- **Relationship A requires Condition B**

  If any product in Relationship A is present in the solution, Condition B is required to be true. If Condition B is false, then all the products in Relationship A are excluded.

- **Condition A requires Product B**

  If Condition A is true in the solution, then Product B is required. If Product B is excluded, then Condition A is required to be false.

- **Condition A requires Relationship B**

  If Condition A is true in the solution, then at least one product in relationship B must be present in the solution. If all the products in Relationship B are excluded, then Condition A is required to be false.

- **Condition A requires Condition B**

  If Condition A is true in the solution, then Condition B must also be true. If Condition B is false, then Condition A is required to be false.

Attribute Conditions

An attribute condition specifies an attribute value and uses it to identify the items in a relationship to which the constraint applies. In the constraint examples below, the attributes are defined on items in relationships within the product with components. You can also define require constraints on the attributes of the product with components itself.

In the constraints examples below, the equals operator is used in the attribute expressions. You can use all the math operators (<, >, and so on) when writing this type of constraint.

- **Product A requires Attribute C = M in Relationship B**

  If Product A is present in the solution, then M is the only value selectable for Attribute C for all items in Relationship B.

- **Relationship A requires Attribute C = M in Relationship B**

  If any product in Relationship A is present in the solution, then M is the only value selectable for Attribute C for all items in Relationship B.

- **Attribute C = M in Relationship A requires Product B**

  If any product with Attribute C = M in Relationship A is present in the solution, then Product B must be present.

- **Attribute C = M in Relationship A requires Relationship B**

  If any product with Attribute C = M in Relationship A is present in the solution, then at least one of the products in Relationship B must be present.

- **Attribute C = M in Relationship A requires Attribute D = P in Relationship B**

  If any of the products with Attribute C = M in Relationship A are present in the solution, then P is the only value selectable for Attribute D in Relationship B.

Quantity Conditions

Quantity conditions compare the quantities of two items. Depending on the session context in which the quantity condition is evaluated, it either returns true/false or is enforced as a constraint.

For example you write the following constraint:

(Product A > Product B) requires Product C

If the user picks Product A so that its quantity is greater than Product B, then Product C is required. In this case, the quantity condition is evaluated as true/false, and Product C is required when it is true.

Contrast this with the following constraint:

Product C requires (Product A > Product B)

When the user picks Product C, the condition (Product A > Product B) is enforced as a constraint. In all further solutions, the quantity of Product A must be greater than the Product B. Also, if the quantity of Product B is greater than Product A in the solution, Product C is excluded.

Other Item Constraints

Several other types of conditions can also be used in require constraints. In some cases these conditions do not make sense when used as the second operand in a require constraint. This is because some conditions are always true (message conditions), or the condition cannot be enforced to be true (linked item conditions).

Table 25 summarizes how to use item constraints

Table 25. Item Constraint Usage in Require Constraints

Item Constraint	First Operand	Second Operand
Attribute Value	Yes	Yes
Consumes	No	Yes
Provides	No	Yes
Excludes	Yes	Yes
Excludes List	Yes	Yes
Require	Yes	Yes
Require List	Yes	Yes
Linked Item Value condition	Yes	No
Message, Recommends	No	Yes
Product Quantity	Yes	Yes
Relationship Quantity	Yes	Yes
Class Quantity	Yes	Yes

Nested Expressions as Conditions

The Require template can itself be used as a condition in other constraints. The most common templates used for writing nested constraints are the Require and Exclude templates. For example, you could write the following configuration constraint:

Product A requires (Product B requires Product C)

The Boolean form of this constraint is as follows:

(NOT A) OR ((NOT B) OR C)

Table 26 shows the truth table for this constraint.

Table 26. (NOT A) OR ((NOT B) OR C)

Row	A	B	C	NOT A	(NOT B) OR C	(NOT A) OR ((NOT B OR C)
1	F	F	F	T	T	T
2	F	F	T	T	T	T
3	F	T	F	T	F	T
4	F	T	T	T	T	T
5	T	F	F	F	T	T
6	T	F	T	F	T	T
7	T	T	F	F	F	F
8	T	T	T	F	T	T

The table lets you analyze what happens when the user picks items. Use the following steps to do this:

1. The engine must return solutions in which all constraints are true. Eliminate any rows where the top-level expression is False. In the table above, eliminate row 7.
2. To determine what happens when the user picks an item, look at all rows that list true for that item. For example, to analyze what happens when the user picks Product A, you would look at rows 5, 6, and 8 in the table above.
3. To determine what happens when a combination of items are picked, look at all the rows that list true for both items at once. For example, to analyze what happens when the user picks both Product A and Product B, you would look at only row 8 in the table above. (Row 7 is not considered since the top-level expression is false.)
4. If only one row has the correct truth conditions, this means the Siebel Configurator engine will return the result shown in that row. For example, look at row 8. This is the only remaining row that lists item A and B as both true. Since this row lists C as true, this means that when both A and B are present, C must be present.

5 If several rows have the condition you are analyzing, look at the truth conditions for each unpicked item in the rows. If they are all true, the unpicked item is required. If they are all false, the unpicked item is excluded. If an unpicked item lists true in some rows and false in others, this means the unpicked item is neither excluded nor required and is available.

The table reveals that the constraint has the following behavior:

- When none of the products are present, the user can pick any of the three, and the other two will not be required.
- If the user picks Product A and B, then Product C is required.

Thus, the constraint's behavior can be summarized this way: when the user picks Product A, the condition "Product B requires Product C" is enforced as a constraint.

Multiple Operands

You can add multiple operands to a require constraint by clicking the Compound Field button when you create the constraint.

For example, you could create the constraint:

Item A requires Item B > 2, Item C < 5

This constraint means the following:

- If Item A is present in the solution, the quantity of Item B must be greater than 2
- If item A is present in the solution, the quantity of Item C must be less than 5
- If Item B cannot be greater than 2 in the solution, Item A is excluded
- If item C cannot be less than 5 in the solution, Item A is excluded

This is the same as writing two constraints:

Item A requires Item B > 2

Item A requires Item C < 5

Require (Mutual) Template

The Require (Mutual) template has the following form:

[Item or Condition] and [Item or Condition] mutually require each other

Use this template when the requires relationship between items or conditions is mutual. For example, Product A requires Product B, AND Product B requires Product A.

When using components as the operands, you can specify global paths for either or both components.

The Boolean equivalent of the Require (Mutual) template is NOT(A XOR B). Table 27 shows the truth table for this template.

Table 27. Truth Table for Require (Mutual)

A	B	A XOR B	NOT(A XOR B)
T	F	True	False
F	T	True	False
F	F	False	True
T	T	False	True

The behavior of the Require (Mutual) template is the same as the Exclude template, except that operands are required instead of excluded.

Set Initial Attribute Value Template

The Set Initial Attribute Value template has the form:

[An attribute] has an initial value of [a number]

The Set Initial Attribute Value template sets the numeric value of an item's attribute at the beginning of a configuration session. The attribute must be of data type Number or Integer. The attribute can have either an enumerated types or range of values type domain.

If attributes' value are set through the Set Initial Attribute Value Constraint template, the user cannot change these values at run time. They can only be changed by Provide and Consume rules.

Setting an attribute value in this fashion brings the attribute value under the control of all its contributors. This means that the attribute value must exactly equal the sum of all its contributors. For example, if during the configuration session, the amount contributed by provide and consume constraints exactly equals the attribute value, users will not be able to change the attribute value. If not, the user can adjust the value, but only if this also adjusts the amount contributed by the provide and consume constraints.

To set the initial value of an attribute that has a non-numeric data type, use the Set Preference template.

This template cannot be used to set the attributes of customizable products that are components in a customizable product. For example, customizable product CP1 has as one of its components customizable product CP2. You cannot use this template to set the values of attributes in CP2.

If the product administrator has set the value of an attribute in the Product Attributes list, this value cannot be overridden by a configuration constraint or by the Siebel Configurator engine.

Set Initial Resource Value Template

The Set Initial Resource Value template has the form:

[A resource] has an initial value of [a number]

The Set Initial resource Value template functions as a provide constraint and sets the numeric value of a resource at the beginning of a configuration session.

Before specifying a resource using this template, items must be present in the solution that contribute (provide or consume) to the resource. During a configuration session, other constraints can increase or decrease the value.

Resource values are under the exclusive control of provide and consume constraints. Users cannot set the value of a resource by entering a value in a selection page.

Set Preference Template

The Set Preference template has the form:

When possible, constrain [an expression] to be true with a [specified priority]

The expression in a preference constraint is enforced as a constraint only if it does not conflict with any other constraint type or with any user selections. The purpose of the Set Preference template is to allow you to create soft constraints that guide the Siebel Configurator engine in producing solutions but which the engine can ignore if needed to avoid conflicts or performance problems.

A key use for preference constraints is to cause a default selection of Item B based on the selection of Item A. This is called a dynamic default. You can set a default dynamically based on a previous user selection. The user can then override the default if desired by choosing a different item than the dynamic default.

For example, you could write the following constraint:

When possible, constrain [Item A requires Item B] to be true with a priority of [0].

When the user picks Item A, the engine will attempt to create a solution containing Item B. However, the engine is free not to include Item B in order to avoid conflicts and performance problems.

If users do not want Item B, they can remove it without creating a conflict. If you had written the constraint as “Item A requires Item B”, Item B would be added when users picks Item A. If user tried to remove Item B, they would receive a conflict message.

Another use for the Set Preference Template is to set or modify the default value for an attribute. To do this, you would write a preference constraint where the expression, is Attribute A = value. The attribute would then be displayed with this value unless overridden by another constraint.

The priority operand in preference constraints determines the order in which multiple preference constraints for an item are evaluated. Preference constraints with priority 0 that apply to a specific item are evaluated first. Those with priority 1 that apply to that item are evaluated next, and so on.

For example, you have written two preference constraints that apply to a specific relationship. PrefConstraint A has a priority of 0. PrefConstraint B has a priority of 1. The Siebel Configurator engine will attempt to add PrefConstraint A to the solution before attempting to add PrefConstraint B.

Here is how the Siebel Configurator engine creates solutions containing preference constraints:

- The engine generates a solution that enforces all constraints and user choices but does not include preference constraints.
- The engine then adds the preference constraints one at a time for each item. It begins by adding the highest priority preference constraints for an item first. Preference constraints with the same priority are added in arbitrary order.
- If the solution remains valid when a preference constraint is added, then the expression in the preference constraint is enforced as a constraint and becomes part of the solution.
- If adding a preference constraint creates a conflict so that the solution fails, the preference constraint is skipped, and its expression is not enforced. The engine then goes on to the next preference constraint.
- Preference constraints are evaluated after all other constraint types and before the engine searches for and sets default attribute values.

Enabling Dynamic Default

To enable dynamic default, set the Dynamic Default UI property for the root product to Y during product definition.

Saved Values for Dynamic Default

Behavior of the Set Preference template depends on whether you have enabled dynamic default.

If the Dynamic Default property is not specified or is set to N, child components that are deleted are reset to the default value specified in the Set Preference template when a user works on a configuration that was saved earlier.

If Dynamic Default property is set to Y, then child components that are deleted are not reset to the default value specified in the Set Preference template when the user works on a configuration that was saved earlier. They are still deleted when the user works on the configuration again.

For example, there is a Set Preference constraint rule that states: Item A requires Item B. When Siebel Configurator is launched, the user selects Item A and the preference constraint triggers the selection of Item B. The user removes Item B by setting its quantity to 0 and then saves the quote. At a later time, the user works on the quote and configures the product again:

- If dynamic default is off, the child component is added to the product, because this is the default in Set Preference, even though the user removed it during the earlier Siebel Configurator session.
- If dynamic default is on, the child component is not added to the product. It is still removed, because the user removed it during the earlier Siebel Configurator session.

Dynamic default applies only to items that are deleted. If the user changes the quantity to a non-zero value or changes any attribute value, the change remains in the next configuration session, and it is not reset to the default value, regardless of whether dynamic default is used.

New Tables for Dynamic Default

If you are upgrading, this list of the new tables used for dynamic default can be helpful to you for data migration and troubleshooting. The following new tables were added to capture the deleted items between configuration sessions:

- S_ASSET_DEL
- S_ORDER_ITM_DEL
- S_QUOTE_ITM_DEL

When the user confirms deleting an item from the configuration session, a new record for the deleted item is inserted in the relevant table. During reconfiguration, the configurator logic checks the tables so that deleted items are not re-included during reconfiguration.

Using the S_ASSET_DEL Table with a Set Preference Constraint

When you deploy a new version of a customizable product that contains a new Set Preference constraint, you may not want this constraint to automatically add a new component to any existing assets when you modify them.

To do this, you must insert the appropriate record into the S_ASSET_DEL table for all existing assets affected by default products.

Observe the following guidelines:

1. Review all existing active assets affected by default products.
2. Add an entry to the S_ASSET_DEL table at the parent asset level, populating the following columns:
   - **ASSET_ID.** Parent asset (not Root) for which the child item, selected by the default product, has been explicitly removed.
   - **PRODUCT_ID.** ID of the product that must not automatically default.
   - **PORT_ID**. This is a reference ID. It belongs to the relationship object of the customizable product that you do not wish to be affected by an automatic default product.

18 Siebel Configurator Rule Assembly Language

This chapter explains how to create configuration constraints using Rule Assembly Language. You can use Rule Assembly Language to enter constraints instead of using constraint templates.

This chapter includes the following topics:

Why Use Rule Assembly Language?

Rule Assembly Language (RAL) is intended for those users who are more comfortable working in a programming environment rather than using templates. Those with experience using this language in previous releases can continue to use it in this release.

In many cases, a combination of RAL and constraint templates can be effectively employed as follows.

1. Use the existing templates to create basic configuration constraints.
2. Create specialized templates and use them to create constraints to handle configuration areas that are similar.
3. Use RAL to create highly complex or unusual constraints not easily handled by templates.

About Rule Assembly Language

All rules in the Rule Assembly Language (RAL) consist of expressions. Expressions consist of an operator and its operands. The number and type of operands depend on the operator. All expressions have the following form:

operator(*A*,*B*...)

For example, the following expression evaluates to the sum of *A* plus *B*.

+(*A*,*B*)

Most operators allow their operands to be expressions. In the expression above, both *A* and *B* can themselves be expressions.

Spaces, tabs, and new-lines are ignored in expressions.

In Rule Assembly Language, a constraint is a list of one or more top-level expressions. A top-level expression is the top-level operator and its associated operands in RAL statements.

For example, in the following statement, +(*A*, *B*) is a sub-expression and is not a top-level expression. The top-level expression is "==" and its operands. So the constraint on all solutions is that the sum of the quantities of A and B must equal the quantity of C in all solutions.

==(+(*A*, *B*), *C*)

A sub-expression is an expression that functions as an operand. Depending on the operator, a sub-expression returns a quantity or a logical true or false. The sub-expression itself is not a constraint.

Creating Constraints Using the Assisted Advanced Constraint Template

A special constraint template is provided for creating constraints using Rule Assembly Language. This template provides a dialog box for picking components, resources, and links. It also provides a list of RAL operators.

When you create a constraint and save it using this template, the Constraints view displays the constraint syntax in the Constraint field. The Constraints view capitalizes the first letter of operator names in the constraint for display purposes only. Operator names are case-sensitive, and the Constraints view stores them in the correct format.

To create a constraint using the Assisted Advanced Constraint template

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Constraints view for the Work Space version.
4. Click New Constraint.

   The Pick a Constraint list appears.
5. In the Pick a Constraint list, select the Assisted Advanced Rule template, and click Continue.

   The Rule Statement and "Insert a" tabs appear.
6. Select operators and arguments from the displayed lists to build a rule.

   The operators list contains all the operators in the Rule Assembly Language. The arguments list changes depending on the operator you select and contains all the items in the customizable product. Use the Compound Field button to create sub-expressions.
7. Click Save Constraint to save the constraint.

   The Save Rule form appears.

8 Fill out the fields in the Save Constraint form, and then click Save. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the constraint. You must use names that help you to locate the constraint using the Find button. For example, consider including the constraint type (excludes, requires and so on) in the constraint name, so you can search the Name field to find groups of constraints having the same constraint type, for example, all the exclude constraints.
Explanation	Enter an explanation of how the constraint works. You must enter explanations that help you to locate the constraint using the Find button. For example, consider including information that uniquely identifies the constraint, such as item names, so you can search the Name and Explanation fields to find a specific constraint.
Rule Statement	Displays the constraint statement that you built.
Start Date	Optionally, specify a start date on which the constraint becomes effective.
End Date	Optionally, specify an end date after which a constraint becomes inactive.
Active	Select this checkbox to activate the constraint, so it is used to compute solutions. Use this feature in the current work space to simulate the behavior of constraints that will have a start date, end date, or both when you release the product. You can also use this feature to deactivate a constraint but retain it in a released version of the product.

9 From the Constraints List menu, select Validate.

This starts a configuration session. Verify that the new rule works correctly.

Creating Constraints Using the Advanced Constraint Template

The Advanced Constraint template is similar to the Assisted Advanced Constraint Template. You can create constraints using Rule Assembly Language with either template. The Advanced Constraint template does not provide a dialog box for picking products, resources, or links. It also does not provide a list of operators.

The Advanced Constraint Template is intended primarily for upgrade users who want to edit the rules in models created in release 6.x.

You must manually enter the path to items when using this template. Table 28 provides examples of paths.

When you create a constraint and save it using this template, the Constraints view displays the constraint syntax in the Constraint field. The Constraints view capitalizes the first letter of operator names in the constraint for display purposes only. Operator names are case-sensitive, and the Constraints view stores them in the correct, format.

Table 28. Examples of Paths

Path	Explanation
@.[Relationship A]([Product SubClass])	All the products in SubClass in Relationship A.
@.[Relationship A]([Product 1]).[Color]	The color attribute of all the instances of Product 1 in Relationship A.
@.[Relationship A].[Color]	The Color attribute of all the products in Relationship 1.
$.[Resource 1]	Resource 1.
$.[Link 1]	Link 1.

Observe the following guidelines when writing paths:

- The @ sign specifies the instance of the product or group of products on which the constraint is defined. Use the @ sign at the beginning of paths that refer to items inside the structure of a customizable product.
- The $ refers to a special object called the basket that is associated with each customizable product. This object maintains a non-hierarchical, flat view of the whole customizable product. Use the $ at the beginning of a path to specify links and resources. Since links and resources are defined for the whole customizable product, they are stored in the basket.

  Upgrade Users. Users upgrading from release 6.x will have rules containing paths that include syntax such as $.[product]. These paths must function normally. However, you must avoid using this syntax to create new rules. This syntax can cause solutions that contain unintended instances of products.
- Use periods (.) to specify the next property in the path. A property can be a relationship name or attribute name. Do not put spaces before or after the period.
- Use parentheses immediately after a relationship name to specify a subset of the items in a relationship. Parentheses act as a filter. The most common use for parentheses is to specify a subclass within the relationship. Do not put a period before the parentheses. For example @.[P]([X]).[Color] refers to the Color attribute of all the products in subclass X within relationship P.

- Enter names exactly as they are displayed in the lists where they were defined. Do not use display names. You can use subclass names to filter products within a relationship even though the subclass names do not display in the Structure view.
- The path syntax always refers to the actual set of items specified in the path, however many instances are present. For example, @.[X]. [Color] refers to the color attribute of all the instances of products actually present in relationship X in a given configuration. A path only refers to actual instances, not the possible instances as defined by cardinality settings.
- If you create a constraint containing a path to location that contains no items, the constraint is ignored until items are present.
- All paths you specify in rules must be unique and unambiguous. For example, if you have multiple relationships with the same name, paths to them may be ambiguous. If this occurs, use the Assisted Advanced Constraint template to write rules. This template uses underlying, unique identifiers to identify items.
- The maximum length of a constraint is 900 characters. The UI allows rules that are longer, but they cannot be stored in the database. Using Oracle's Web Tools, you can revise the UI to enforce the 900 character limit. See "Enforcing the Field Length for Entering Advanced Rules" on page 409.

To create a constraint using the Advanced Constraint Template

1. Navigate to the Administration - Product screen.
2. Select and lock the desired customizable product or product class.
3. Navigate to the Constraints view for the Work Space version.
4. Click New Constraint.

   The Pick a Constraint list appears.
5. In the Pick a Constraint list, select the Advanced Rule template, and click Continue.

   The Rule Statement and "Insert a" tabs appear.
6. Select operators and arguments from the displayed lists to build a constraint.

   The operators list contains all the operators in the Rule Assembly Language. The arguments list changes depending on the operator you select and contains all the items in the customizable product. Use the Compound Field button to create sub-expressions.
7. Click Save Constraint to save the constraint.

   The Save Rule form appears.

8 Fill out the fields in the Save Constraint form, and then click Save. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the constraint. You must use names that help you to locate the constraint using the Find button. For example, consider including the constraint type (excludes, requires and so on) in the constraint name, so you can search the Name field to find groups of constraints having the same constraint type, for example, all the exclude constraints.
Explanation	Enter an explanation of how the constraint works. You must enter explanations that help you to locate the constraint using the Find button. For example, consider including information that uniquely identifies the constraint, such as item names, so you can search the Name and Explanation fields to find a specific constraint.
Rule Statement	Displays the rule statement that you built.
Start Date	Optionally, specify a start date on which the constraint becomes effective.
End Date	Optionally, specify an end date after which a constraint becomes inactive.
Active	Select this checkbox to activate the constraint, so it is used to compute solutions. Use this feature in the current work space to simulate the behavior of constraints that will have a start date, end date, or both when you release the product. You can also use this feature to deactivate a constraint but retain it in a released version of the product.

9 From the Constraints List menu, select Validate.

This starts a configuration session. Verify that the new rule works correctly.

Managing Constraints Written in Rule Assembly Language

Use the same procedures for copying, editing, and deleting constraints written in Rule Assembly Language that you use for constraints written using constraint templates.

You can modify the Constraints list to display the RAL version of each constraint you create using constraint templates. This is a useful way to learn RAL and to understand more fully how constraint templates work. To modify the Constraints list, see "Displaying RAL in the Constraints View" on page 410.

About Specifying Data in Rule Assembly Language

This topic describes how to specify numbers, strings, names, data types, and property types in Rule Assembly Language.

Numbers

You can use both integers and floating point numbers. Floating point numbers contain a decimal point.

- Example of integers: `1, 100, -239`
- Example of floating point: `3.14, 1.0, 10.567`

Strings

Enclose strings in double quotes. For example:

```
"Parker Data Systems recommends a DSL modem"
```

White space in a string is treated as a character. Use a back-slash (\) as an escape character to include double quotes or a back-slash in a string. For example:

```
"Install these fonts in C:\\psfonts on your system"
```

If you put quotes around a number or anything else, it is treated as a string.

Links

Links store data extracted from Siebel databases. Links can also store the value of specific system variables. Links can be used only to define conditions. Enclose link names in square brackets.

About Operators in Rule Assembly Language

In expressions, operators define operations or relationships between operands. Operator names are case sensitive. For example, `Req(A, B)` is not the same as `req(A, B)` and will result in a syntax error.

Most operator names are entirely lowercase. However, a few contain capital letters and are noted in later topics.

Some operators expect logical operands. Others expect numeric operands. When an operand is of a type different than the operator expects, the Siebel Configurator engine forces the operand to the correct type.

- When integers are used where floating point is expected, integers are converted to their double-precision floating point equivalent.
- When floating point numbers are used where integers are expected, floating point numbers are rounded to their nearest integer value.
- Numbers greater than zero are interpreted by logical operators as true.
- Numbers less than or equal to zero are interpreted by logical operators as false.
- When used as numeric operands, true is 1 and false is 0.

The different types of Rule Assembly Language operators are described in the following topics:

- "Data Operators in Rule Assembly Language" on page 285. Data operators support expressions involving data that originates elsewhere in the Siebel Business Application.
- "Boolean Operators in Rule Assembly Language" on page 285. Boolean operators take logical operands and return logical results. For example: and(A, B).
- "Comparison and Pattern Matching Operators in Rule Assembly Language" on page 288. and pattern matching operators take numeric operands and return logical results. For example: >(A, B).
- "Arithmetic Operators in Rule Assembly Language" on page 289. Arithmetic operators take numeric operands and return numeric results. They provide math operations such as addition and subtraction.
- "Attribute Operators in Rule Assembly Language" on page 291. Attribute operators do comparisons and particular math operations on attribute values.
- "Conditional Operators in Rule Assembly Language" on page 293. Conditional operators provide conditional logical and numerical relationships, such as if-then-else.
- "Special Operators in Rule Assembly Language" on page 293. Special operators interpret their operators in a special way. Some provide access to the configuration session: for example, to signal messages or retrieve property values. They also provide binding and iteration services.
- "Customizable Product Access Operators in Rule Assembly Language" on page 297. Customizable product access operators allow you to obtain information about other areas of the customizable product.

Data Operators in Rule Assembly Language

Use the following operators, shown in Table 29, when working with Siebel data types.

Table 29. Siebel Data Operators

Operator	Syntax	Properties
Number	`Number(A)`	Converts the operand to a number. Operand can be an expression.
String	`String(A)`	Converts the operand to a string. Operand can be an expression.
Date	`Date(A)`	Converts string to a date. Operand can be an expression.
Time	`Time(A)`	Converts the operand to a time. Operand can be an expression.
UtcDateTime	`UtcDateTime(A)`	Converts the operand to the UTC date and time. Operand can be an expression.
DateTime	`DateTime(A)`	Converts the operand to a date and time. Operand can be an expression. Not recommended for use. Use UtcDateTime instead.
Currency	`Currency(A)`	Converts the operand to currency. Operand can be an expression.
Phone	`Phone(A)`	Converts the operand to a phone number. Operand cannot be an expression. Can be used only with Equals (==) and Not Equal To (!=) operators to compare two phone numbers.

Boolean Operators in Rule Assembly Language

When you specify this type of operator, the Siebel Configurator engine interprets it as true when the item is in the solution, and as false if the item is not in the solution. If you specify a resource or an arithmetic sub-expression as an operand, the Siebel Configurator engine interprets it as true if the expression is greater than zero, and false if the expression is less than or equal to zero.

The requires operator (req()) is an example of this type of operator. The Siebel Configurator engine interprets the following constraint to mean item A requires item B. In other words, if A is in the solution, B must be in the solution.

```
req([A], [B])
```

The Siebel Configurator engine does not interpret this constraint to mean the current quantity of item A requires the same quantity of item B. That constraint would be written as follows:

```
==([A], [B])
```

The Boolean operators are shown in Table 30.

Table 30. Boolean Operators

Operator	Syntax	Properties
Not	!(*A*)	Logical negation. True when *A* is false and false when *A* is true. *A* can be an item or sub-expression.
Requires	req(*A*, *B*)	*A* implies *B*. False only when *A* is true and *B* is false. *B* does not require *A*. However, excluding *B*, excludes *A*. *A* and *B* can be items or sub-expressions.
Excludes	excl(*A*, *B*)	*A* excludes *B* and *B* excludes *A*. Same as "not both *A* and *B*." False only when *A* and *B* are both true. *A* and *B* can be items or sub-expressions.
And	and(*A*, *B*)	Both *A* and *B*. True only when both *A* and *B* are true. When used as a top-level constraint, means that only solutions where both *A* and *B* are true are allowed. *A* and *B* can be items or sub-expressions.
Inclusive Or	or(*A*, *B*)	Either *A* or *B* or both. False only when both *A* and *B* are false. *A* and *B* can be items or sub-expressions.
Exclusive Or	xor(*A*, *B*)	*A* or *B* but not both. *A* and *B* must have opposite truth states. False when *A* and *B* are either both true or both false. *A* and *B* can be items or sub-expressions.
Logically Equivalent	eqv(*A*, *B*)	*A* requires *B* and *B* requires *A*. True only when *A* and *B* are either both true or both false. *A* and *B* can be items or sub-expressions.

Table 31 provides the truth-state definition of how the Boolean operators work. The first two columns contain the operands A and B. These could be items or arithmetic expressions, in which case A>0 means that A is in the solution.

Table 31. Truth Table for Boolean Operators

A >0?	B>0?	req(A, B)	excl(A, B)	and(A, B)	or(A, B)	xor(A, B)	eqv(A, B)
T	T	T	F	T	T	F	T
T	F	F	T	F	T	T	F
F	T	T	T	F	T	T	F
F	F	T	T	F	F	F	T

More on the Requires Operator

The requires operator is not an incremental add. For example, you write the constraint req(A, B). If the user picks A and there are no B's in the solution, the Siebel Configurator engine will add at least one B. The next time the user picks A, the engine does not add another B.

If you want to add a B each time an A is added, use the inc() operator.

More on the Logical Equivalence Operator

As a top level constraint, the logical-equivalence operator (eqv()) creates a mutually-requires relationship between its operands. The operands can be either items or sub-expressions. For example eqv([A], [B]) means than if item A is in the solution, then at least one item B must be in the solution. Also, if item B is in the solution, then at least one item A must be in the solution. Note that the relationship between [A] and [B] is noncumulative. (Use the inc() operator to create cumulative-requires relationships.)

For example, the following constraint states that if the quantity of [A]>2, then [B] is required.

```
eqv(>([A],2),[B])
```

This expression constrains the solution as follows:

- If there are more than two A's in the solution, there must be at least one B.
- If there cannot be any B's in the solution, there cannot be any more than two A's.
- If B is in the solution, there must be more than two A's.
- If A is limited to two or less, B is excluded.

More on the Excludes Operator

As a top-level constraint, the excludes operator (excl(A, B)) creates a mutually exclusive relationship. The expression excl([A], [B]) means that if item A is in the solution, item B cannot be in the solution. It also means that if item B is in the solution, item A cannot be in the solution.

The exclude constraint can also be used with sub-expressions. For example, the following constraint excludes item B when item A's quantity is greater than 2. It also prevents item A from being greater than 2 when item B is in the solution.

```
excl(>([A],2),[B])
```

Multiple Operands for Require and Exclude Operators

You can use multiple operands in requires and exclude constraints. For example, you could write the following constraint:

```
excl([A],[B],[C])
```

This syntax is interpreted by the Siebel Configurator engine as if you had written two constraints:

```
excl([A],[B])
```

```
excl([A],[C])
```

In other words, [A] excludes *both* [B] and [C]. Note that [A] is the first operand in both the constraints. Siebel Configurator takes the first operand and creates expressions between it and each remaining operands.

This works the same way for require constraints:

```
req([A],[B],[C])
```

This syntax is interpreted by the Siebel Configurator engine as if you had written two constraints:

```
req([A],[B])
```

```
req([A],[C])
```

In other words, [A] requires *both* [B] and [C]. There is no limitation on the number of operands you can use in this type of expression.

Comparison and Pattern Matching Operators in Rule Assembly Language

Comparison operators expect numeric operands. This means when you specify an item or arithmetic expression, the Siebel Configurator engine uses the quantity of the item or the value of the expression. These operators produce a logical result. This means the operator evaluates and compares the quantities of the operands and returns a true or false result.

The greater-than operator (>) is an example. The Siebel Configurator engine interprets the following top-level constraint to mean the quantity of item A must be larger than the quantity of item B in the solution. The Siebel Configurator engine enforces this constraint by adjusting the quantity of A or B as needed to make sure that the constraint is always true.

```
>([A],[B])
```

When used as sub-expressions, comparison operators return true or false. For example, if the quantity of item A in the solution is not greater than the quantity of item B, the previous example returns false. This is then acted on by the associated top-level expression.

Pattern matching operators compare two strings. You can test whether the strings are a match or a mismatch. Pattern matching operators return true or false.

Comparison operators are shown in Table 32.

Table 32. Comparison and Pattern Matching Operators

Operator	Syntax	Properties
Greater than	>(*A*, *B*)	*A* and *B* can be items or sub-expressions.
Not less than	>=(*A*, *B*)	*A* and *B* can be items or sub-expressions.
Equals	==(*A*, *B*)	*A* and *B* can be items or sub-expressions.
Not equal to	!=(*A*, *B*)	*A* and *B* can be items or sub-expressions.
Not greater than	<=(*A*, *B*)	*A* and *B* can be items or sub-expressions.

Table 32. Comparison and Pattern Matching Operators

Operator	Syntax	Properties
Less than	<(*A*, *B*)	*A* and *B* can be items or sub-expressions.
Selected	sel(*A*)	Returns true if *A* is positive, false if *A* is not. Same as >(*A*, 0). If used as top-level expression, means that *A* must be in the solution in any quantity. *A* can be an item or sub-expression.

Multiple Operands for Comparison Operators

You can use multiple operands in constraints containing comparison operators. For example, you could write the following constraint:

```
>([A],[B],[C])
```

This syntax is interpreted by the Siebel Configurator engine as if you had written two constraints:

```
>([A],[B])
```

```
>([A],[C])
```

Note that [A] is the first operand in both the constraints. Siebel Configurator takes the first operand and creates expressions between it and each remaining operands. There are no limitations on the number of operands you can use in this type of expression.

Note that the expression are interpreted by the Siebel Configurator engine as pairs that always include the first operand. This type of expression does not create implied constraints between other operands. For example, you write the following constraint:

```
!=([A],[B],[C])
```

This constraint expression is interpreted as A !=B and A!=C. It does not imply that B !=C.

Arithmetic Operators in Rule Assembly Language

Arithmetic operators expect numeric operands and produce a numeric result. They are most frequently used in sub-expressions. The following top-level expression means that the quantity of item C in the solution must be the same as the sum of the quantities of items A and B.

```
==(+([A],[B]),[C])
```

Assuming no other constraints on item A, B, or C; if you add A or B to the solution, then C will be added as well to match the sum. If you add a large number of C's, the Siebel Configurator engine will add A and B in arbitrary quantities so that their sum equals the amount of C.

When used in sub-expressions, these operators must return a numeric result. If a sub-expression returns a logical result, true is interpreted as a 1, and false is interpreted as a 0. In the example above, if B is an expression that returns the logical result true, then the expression is equivalent to the following:

```
==(+([A],1),[C])
```

Arithmetic operators are shown in Table 33.

Table 33. Arithmetic Operators

Operator	Syntax	Properties
Addition	+(*A*, *B*)	Sum of *A* and *B*. *A* and *B* can be items or sub-expressions. Result is floating point if A or B is floating point.
Subtraction	-(*A*, *B*)	Subtracts *B* from *A*. *A* and *B* can be items or sub-expressions. Result is floating point if *A* or *B* is floating point.
Negation	-(*A*)	Additive inverse of *A*. Uses only one operand. *A* can be an item or expression.
Multiplication	*(*A*, *B*)	Product of *A* and *B*. Result is floating point if *A* or *B* is floating point. *A* and *B* can be items or sub-expressions.
Division	/(*A*, *B*)	Quotient of *A* divided by *B*. Truncates ratio to integer if both *A* and *B* are integers. Result is floating point if *A* or *B* is floating point. *A* and *B* can be items or sub-expressions.
Modulo	%(*A*, *B*)	Remainder of *A* divided by *B*. For example, %(1900, 72) results in 28. If *A* or *B* is floating point, the value is first rounded to the nearest integer; then the remainder is computed as for integers. *A* and *B* can be items or sub-expressions.
Minimum	min(*A*, *B*)	Result is the smaller of *A* and *B* and is floating point if *A* or *B* is floating point. *A* and *B* can be items or sub-expressions.
Maximum	max(*A*, *B*)	Result is the larger of *A* and *B* and is floating point if *A* or *B* is floating point. *A* and *B* can be items or sub-expressions.

The following operators, shown in Table 34, also take numeric arguments and produce numeric results. Use them to control numeric accuracy or change numeric characteristics.

Table 34. Additional Arithmetic Operators

Operator	Syntax	Properties
Quantity	qty(*A*)	Result is the quantity of *A* rounded to nearest integer. For example, if *A* is 6.7, returns 7. If *A* is 6.3, returns 6. *A* can be an item or sub-expression. Useful only with resources.
Integer	int(*A*)	Truncates *A* down to an integer. For example, if operand is 6.7, returns 6. *A* can be an item or sub-expression.
Float	flo(*A*)	Converts *A* to floating point. Same as multiplying operand by 1.0. *A* can be a sub-expression. Not useful with resources.
Absolute value	abs(*A*)	Returns the absolute value of *A*. *A* can be an item or sub-expression.
Sign test	sgn(*A*)	Returns -1 if the quantity of *A* <0, 0 if *A*=0, 1 if *A*>0. *A* can be a sub-expression.

Attribute Operators in Rule Assembly Language

Rule Assembly Language includes special operators for doing comparisons and particular math operations on attribute values. These operators extract information about the attributes of all the products that have been selected in a relationship. For example, you can determine the number of relationship items that have been selected that have an attribute value greater than a specified amount.

Attribute Comparison Operators

These operators return the number of relationship items that have been selected for which the comparison is true. For example, you can use `numAttr>` to find out how many items with Length greater than 5 feet in a relationship have been selected.

The operators count all the items selected from the relationship, not the number of different items. In the preceding example, if the user selects two of the same item and enters a length greater than 5 feet, the `numAttr>` operator will return 2.

The operators take two arguments, A and B. Argument A is the full path from the root of the product to the attribute. Argument B is the comparison value. This value can be of type Integer, Number, Date, or Time. Type DateTime is not supported. Argument B can also be a sub-expression that resolves to one of these data types.

In addition, for the `numAttr==` and `numAttr!=` operators, argument B can be a text string.

Use attribute comparison operators to create subexpressions that form conditions. Attribute comparison operators are shown in Table 35.

Table 35. Attribute Comparison Operators

Operator	Syntax	Properties
Greater than	`numAttr>(`*A, B*`)`	Returns the number of items in the relationship for which the value of attribute *A* is greater than *B*.
Not less than	`numAttr>=(`*A, B*`)`	Returns the number of items in the relationship for which the value of attribute *A* is greater than or equal to *B*.
Equals	`numAttr==(`*A, B*`)`	Returns the number of items in the relationship for which the value of attribute *A* is equal to *B*.
Not equal to	`numAttr!=(`*A, B*`)`	Returns the number of items in the relationship for which the value of attribute *A* is not equal to *B*.

Table 35. Attribute Comparison Operators

Operator	Syntax	Properties
Not greater than	`numAttr<=(A, B)`	Returns the number of items in the relationship for which the value of attribute *A* is less than or equal to *B*.
Less than	`numAttr<(A, B)`	Returns the number of items in the relationship for which the value of attribute *A* is less than or equal to *B*.

You can use the numAttr operators to create “any/all” conditions in constraints involving attributes:

- You create the condition “any instance of products in R has attribute A=X” as follows:

  `>=(numAttr==(@.[R].[A], X), 1)`

- You create the condition “all instances of products in R have attribute A=X” as follows:

  `==(numAttr==(@.[R].[A], X), @.[R])`

For example, you want to write the constraint, when all the instances of products in P have attribute A=X then exclude any instance of products in Q that have attribute B =Y. You would write this constraint as follows:

```
excl(==(numAttr==(@.[P].[A], X), @[P]),
>=(numAttr(==(@.[Q].[B], Y), 1)))
```

Attribute Arithmetic Operators

The arithmetic operators allow you to determine the maximum and minimum value of an attribute for items that have been selected in a relationship. You can also sum the values of the attributes.

The operators take one argument, which is the path to the attribute. Attributes can be of type Number, Integer, Date, or Time. DateTime and Text are not supported. If the type is Date or Time, `minAttr` returns the time or date closest to the present. The `maxAttr` operator, returns the time or date furthest from the present.

Use attribute arithmetic operators to create subexpressions that form conditions. Attribute arithmetic operators are shown in Table 36.

Table 36. Attribute Arithmetic Operators

Operator	Syntax	Properties
Minimum	`minAttr(A)`	For items selected from a relationship, returns the smallest value for attribute *A*.

Table 36. Attribute Arithmetic Operators

Operator	Syntax	Properties
Maximum	`maxAttr(A)`	For items selected from a relationship, returns the largest value for attribute *A*.
Sum	`sumAttr(A)`	Returns the sum of attribute A for all items selected from a relationship.

Conditional Operators in Rule Assembly Language

Conditional logic operators are shown in Table 37.

Table 37. Conditional Operators

Operator	Syntax	Properties
Logical conditional	`if (A, B, C)`	If *A* then *B*, else *C*. If *C* is not specified, it defaults to true (1).
Numeric conditional	`?(A, B, C)`	If *A* then *B*, else *C*. If *C* is not specified, it defaults to false (0). This means the expression returns false if *A* is false and *C* is not specified. Rarely used.

Special Operators in Rule Assembly Language

Special operators provide functions that manipulate the Siebel Configurator engine directly, rather than the underlying customizable product. They also provide defined types of access to the components of a customizable product.

The inc() operator is used to implement provide and consume constraints. It has two important characteristics:

- It returns a null value.
- When it is evaluated in a subexpression, this causes the Siebel Configurator engine to contribute the value requested.

This means that writing constraints with the inc() operator as a subexpression based on a condition do not work. The contribution will always occur. For example, you write the following constraint.

```
req(X, inc(Y, Z))
```

The Siebel Configurator engine will contribute the value of Y to Z, regardless of whether X is present in the solution. This is because the Siebel Configurator must evaluate the two arguments to req() before determining what action to take. Evaluating the inc() argument causes the engine to contribute the value of Y to Z.

In addition, regardless of the value of X, the inc() argument always evaluates to null, which makes the constraint meaningless.

To write constraints that contribute conditionally, use the numeric conditional operator, ?().

Special operators are shown in Table 38 on page 294.

Table 38. Special Operators

Operator	Syntax	Properties
Constraint	con(*A*)	Makes *A* a constraint. Returns no result. *A* can be an item or sub-expression. Use to make sub-expressions into constraints. Redundant for top-level expressions. Rarely used.
Increment	inc(*A*,*B*)	Each A requires a B. *A* can be a number, item, or sub-expression but must resolve to a number. *B* must be an item and cannot be a sub-expression. inc(1, *B*) increments the quantity of *B* in the solution by 1. Use inc(*A*,*B*), where *A* and *B* are items, to implement cumulative require constraints.
Message	msg(*A*) "string"	Causes message signal when *A* is true (selected). User-defined message displays in user message area. Returns no result. *A* can be an item or sub-expression.
Check	chk(*A*) "string"	Causes a message-signal when *A* is false (not selected). User-defined message displays in user message area. Returns no result. *A* can be an item or sub-expression.
Preference	prefer(*expression*, *priority*)	The *expression* is enforced as a constraint only if it does not conflict with any other constraint type or with any user selections. The *priority* determines the order in which multiple preference constraints for an item are evaluated. Preference constraints with priority 0 that apply to a specific item are evaluated first. Those with priority 1 that apply to that item are evaluated next, and so on.

Table 38. Special Operators

Operator	Syntax	Properties
With Members	`withMembers(`*A*`,`*B*`)`	Enforce the constraint *B* only on instances in *A*. This operator allows you to move the context for enforcement of a constraint from the root of a customizable product to a location within it. For example, *A* can be a path to a relationship within the customizable product. This causes the constraint *B* to be enforced only within that relationship. *A* can be a relationship, a class within a relationship, or a single product. *B* can be any expression or constraint. In *B*, the path must be specified relative to the path in *A*. For example, if *A* specifies relationship R1, the paths to items in *B*, must be specified relative to R1, not the product root. When specifying items in *B*, do not include any part of the path specified in *A*. When adding items to *B* using the pick applet in the Assisted Advanced Constraint template, you must manually edit the item paths in *B* to make them relative to the path specified in *A*.
With Tuples	`withTuples (((A, B, C), (D,E F), ...), ruleA(%1, %2, %3,...),...)`	This operator allows you to specify multiple sets of operands for one or more constraints. The first operand in a group is assigned to the variable %1, the second to %2, and so on. You can specify more than one constraint.

More on withTuples

The withTuples operator lets you provide more than one set of operands to a constraint. It has the following syntax:

```
withTuples (((A, B, C),(D,E F),...), ruleA(%1,%2,%3,...),...)
```

For example, you have the following two constraints:

```
req(and([A],[B]),excl([C],[G]))
```

When both A and B are present, C and G exclude each other

```
req(and([D],[E]),excl([F],[G]))
```

When both D and E are present, F and G exclude each other.

The previous two constraints can be considered one required constraint that has two sets of operands. The withTuples operator lets you write the constraint in this fashion. This makes constraint maintenance easier. If the operands change, you can edit them in one location, rather than having to locate all the constraints in which they appear. Here is one way to combine the two constraints using withTuples:

```
withTuples((([A],[B],[C]),([D],[E],[F])),req(and(%1,%2),
excl(%3,[G]))
```

Notice that each group of operands is enclosed in parentheses. Also notice that the whole section where the operands are specified is itself enclosed in parentheses.

You can also use the withTuples operator to specify the operands for multiple constraints. For example, you have the following two constraints:

```
req(and([A],[B]),[C])
inc([C], [Resource1])
```

If both A and B are present, C is required, and contribute the value of C to Resource1.

These two constraints are different but they make use of the same operands. You could use the withTuples operator to show that these two constraints use the same operands as follows:

```
withTuples((([A],[B],[C])),req(and(%1,%2),%3),inc(%3,[Resource1]))
```

NOTE: For more information, see Article ID 2249407.1 on My Oracle Support.

More on withMembers

The withMembers operator shifts the context for application of a constraint from the root of the customizable product to a specified location within the product. This means the constraint applies only to the items in the specified path, rather than wherever these items appear in the product.

The withMembers operator adds no other functionality and does not alter the function of other operators.

For example, you define Relationship A that contains only the customizable product desktop PC. The desktop PC includes two relationships: CPU and Hard Drive. You want to write the constraint CPU requires Hard Drive and enforce the constraint for each instance of the desktop PC. If the user adds a CPU and hard drive to a desktop PC but later removes the hard drive, you want the user to receive a configuration error for that PC.

You would write this constraint as follows:

```
withMembers(@.Relationship A, req(@.CPU, @.Hard Drive))
```

This constraint enforces “CPU requires Hard Drive” separately on each instance of desktop PC in Relationship A. All the desktop PCs from Relationship A must have a hard drive if they have a CPU. The constraint is not enforced for PCs that appear elsewhere in the customizable product other than Relationship A.

Customizable Product Access Operators in Rule Assembly Language

Customizable Product access operators, shown in Table 39, allow you to obtain information about other areas of the customizable product. For example, you can obtain the name of an item's parent.

Table 39. Customizable Product Access Operators

Operator	Syntax	Properties
Product root	`root()`	Result is root of the customizable product. Takes no arguments.

Examples of Constraints Using Rule Assembly Language

This topic contains examples of how to use the Siebel Configurator Rule Assembly Language to create constraints. Some of the constraints show item names consisting of both the product name and its configuration ID (Cfg ID).

Basic Constraints

The following table shows how create basic constraints using Rule Assembly Language.

Constraint Type	Advanced Rule Language
A requires B noncumulatively	`req(A, B)`
A requires B cumulatively	`inc(A,B)`
A excludes B	`excl(A,B)`
A provides the amount B to C	`inc(*(A,B),C)`
A consumes the amount B from C	`inc(*(A,-(B)),C)`
A's minimum quantity is B, enforced	`>=(A,B)`
A's maximum quantity is B, enforced	`<=(A,B)`
A recommends B	`rec(req(A,B))`

Boolean and Comparison Operators

The following table shows how to create constraints using Boolean and comparison operators.

Constraint Type	Advanced Rule Language
A AND B =C	==(and(*A,B*),*C*)
A OR B = C	==(or(*A,B*),*C*)
NOT (A = B)	xor(*A,B*)
(A < B) requires C	req(<(*A,B*),*C*)
(A <= B) requires C	req(<=(*A,B*),*C*)
(A = B) requires C	req(==(*A,B*),*C*)
(A != B) requires C	req(!=(*A,B*),*C*)
(A >=B) requires C	req(>=(*A,B*),*C*)
(A >B) requires C	req(>(*A,B*),*C*)
(A + B) contributes to C	inc(+(*A,B*),*C*)
(A - B) contributes to C	inc(-(*A,B*),*C*)
(A * B) contributes to C	inc(*(*A,B*),*C*)
(A/B) contributes to C	inc(/(*A,B*),*C*)
(A MIN B) contributes to C	inc(min(*A,B*),*C*)
(A MAX B) contributes to C	inc(max(*A,B*),*C*)

Constraint Template Translations

Table 40 shows examples of the constraint templates translated into Rule Assembly Language.

Table 40. Constraint Template Translations to Rule Assembly Language

Template	RAL Equivalent	Explanation
Constrain	con(>(Item A),1))	Constrain (quantity of Item A> 1) to be true.
Constrain Conditionally	if(>(Item A), 1), >= (Item B), 2), excl(Item C, Item D))	When (quantity of Item A> 1)(quantity of Item B >= 2), otherwise Selection of Item C excludes selection of Item D.
Constrain Product Quantity	==(Item A, 2)	The quantity of Item A = 2.

Table 40. Constraint Template Translations to Rule Assembly Language

Template	RAL Equivalent	Explanation
Consume	inc(*(Item A), -(1)), Resource B)	Each Item A consumes 1 from Resource B.
Exclude	excl(Item A, Item B)	Selection of Item A excludes selection of Item B.
Exclude List	excl(Item A, Item B, Item C))	Item A excludes (selection of Item B, selection of Item C).
Message	msg(>(Item A, 1))	When (Item A > 1), display this constraint's description.
Preference	prefer(Item A, 1)	When possible, constrain Item A is selected to be true with a priority of 1.
Provide	inc(*(Item A), 1), Resource B)	Each Item A provides 1 to Resource B.
Recommends	chk(req(Item A, Item B), "Item B is recommended")	Selection of Item A recommends selection of at least one Item B by displaying "Item B is recommended".
Requires	req(Item A, Item B)	Item A requires Item B.
Requires (Mutual)	eqv(Item A, Item B)	Item A requires Item B AND Item B requires Item A.
Constrain Attribute Value	==(@.[Class A] ([Item B]).[Day], "Monday")	The attribute Class A “Item B” Attribute C = Monday.

19 Siebel Configurator Scripts

This chapter explains how to use the Scripts view to enhance the behavior of Siebel Configurator. To use scripting events and methods you must be familiar with Siebel Visual Basic or Siebel eScript programming. This chapter includes the following topics:

About Siebel Configurator Scripts

A script is a Siebel Visual Basic or Siebel eScript program that runs when a predefined event to which it is assigned occurs during a configuration session. These events occur at specific points when opening or closing a session or processing a user request. You can also place scripts in a declarations area for use by other event-based scripts.

A script can refer to any product that has been added from the product table to a product with components. These are called component products. Scripts can also be used to set attribute values of component products and of the product with components, and of product classes. Scripts cannot be used to refer to relationships or links.

NOTE: Scripting is not supported on the Product Administration business object. It is recommended that you use the Siebel Business Application for product administration, and create scripts to work with those products.

Scripts have full access to the Siebel API. You can use scripts to call Siebel business components and to provide information to back-end databases through Siebel business services. For a description of basic API methods, see Chapter 26, "Siebel Configurator API Reference."

In addition, several special Siebel Configurator methods are described in this chapter. These can be included in scripts and allow you to obtain information about the current solution. You can also use them to submit requests to the Siebel Configurator engine. For example, you can write a script and insert it into an event that is called each time an item quantity changes in the current solution. When the user selects or removes an item, this event is called and your script runs. The script can use Siebel Configurator methods to determine the amount of the item or of other items in the current solution and can submit a request to the Siebel Configurator engine to modify item amounts. The script could also perform special computations or update an external application based on information obtained from the Siebel Configurator methods.

The items and rules that comprise the product are called the declarative portion. The declarative portion cannot be modified by scripts. For example, you cannot add or delete rules from a customizable product using scripts.

Scripts are intended to add additional behaviors or features and can be used in several ways:

- **Arithmetic functions.** You can use scripting to perform more complex mathematical operations that cannot feasibly be created using configuration rules. For example, if you are configuring a solution for a chemical manufacturing process, you can use calculations based on viscosity, average ambient temperature, and desired throughput to arrive at the correct configuration of pumps and other equipment.
- **External program calls.** You can use scripts to make calls to external programs. For example, you can pass information about the current session to external programs.
- **Accessing Siebel objects.** You can use all the standard features of Siebel VB and Siebel eScript, including reading and editing Siebel databases and calling Buscomps, BusObjects, and other Siebel objects.

About Siebel Configurator Script Processing

You can write event scripts or declarations scripts. Declarations scripts contain methods that can be called by event scripts and other declarations scripts.

A script instance is created at the beginning of the associated event and destroyed at the end of the script execution. Variables defined in the declarations section of the script are meaningful only during script execution and do not persist after the script exits. For example, if a script is called because an item has changed, its variable values do not persist. The next time an item changes and the script runs again, the values of the variables from the first script execution are not available.

The events for Siebel Configurator script processing are listed in the following paragraphs.

EV_CFG_INSTINITIALIZED

This event is generated once for each session after the root customizable product is instantiated and before any user requests are accepted.

It triggers the execution of Cfg_InstInitialize (RootProd) of each product present in the current solution.

The input parameter of the script is always the name of Root CP.

EV_CFG_INSTCLOSED

This event is generated once for each session after the user clicks Done to end a configuration session. After this event is generated, no further processing by the Siebel Configurator engine occurs.

It triggers the execution of Cfg_InstPostSynchronize(RootProd) of the root CP. The input parameter of the script is always the name of the root CP.

EV_CFG_ONCONFLICT

This event is generated when the Siebel Configurator engine encounters a conflict.

It triggers the execution of Cfg_OnConflict(Explanation, Resolution) of the root CP.

The input parameters of the script are:

- **Explanation.** An input string, containing a system message explaining the conflict.
- **Resolution.** An output string, *UndoLastRequest* or *RemoveFailedRequests*, instructing the application to undo the last changes or to make the necessary modifications to resolve the conflict and preserve the last changes.

EV_CFG_CHILDITEMSELECTED

This event is generated before the Siebel Configurator engine computes a new solution.

It triggers the execution of Cfg_ChildItemSelected (SelectedItem) for the root CP.

The input parameter of the script is a PropSet, containing all the products whose quantities are about to change as elements:

```
<SelectedItem ObjName= “objname” OldQty= “oldqty” NewQty= “newqty”>
```

EV_CFG_CHILDITEMCHANGED

This event is generated internally after the Siebel Configurator engine computes a new solution when detecting EV_CFG_ITEMCHANGED. It is issued together with EV_CFG_ATTRIBUTECHANGED.

It triggers the execution of Cfg_ChildItemChanged(ChangedItem) of the root CP.

The input Parameter of the script is a PropSet, containing all the products whose quantities have changed as elements:

```
<ChangedItem ObjName= “objname” OldQty= “oldqty” NewQty= “newqty”>
```

EV_CFG_ITEMCHANGED

This event is generated after the Siebel Configurator engine computes a new solution.

It triggers the execution of Cfg_ItemChanged (ProdName, OldQty, NewQty) of each product whose quantities have changed.

The input parameters of the script are:

- **ProdName.** The name of the product.
- **OldQty.** The quantity prior to the request.
- **NewQty.** The resulting quantity after the request.

EV_CFG_ATTRIBUTESELECTED

This event is generated before the Siebel Configurator engine computes a new solution.

It triggers the execution of Cfg_AttributeSelected(ChangedItem) of each product whose attribute values are about to change.

The input parameter of the script is a PropSet containing the changed attributes as elements:

```
<strObjID>
<strAttrName OldVal = "oldval" NewVal = "newval" >
<strAttrName OldVal = "oldval" NewVal = "newval" >
<strAttrName OldVal = "oldval" NewVal = "newval" >
<strObjID>
```

EV_CFG_ATTRIBUTECHANGED

This event is generated after the Siebel Configurator engine computes a new solution.

It triggers the execution of Cfg_AttributeChanged(ChangedItem) of each product whose attribute values have changed.

The input parameter of the script is a PropSet containing the changed attributes as elements:

```
<strObjID>
<strAttrName OldVal = "oldval" NewVal = "newval" >
<strAttrName OldVal = "oldval" NewVal = "newval" >
<strAttrName OldVal = "oldval" NewVal = "newval" >
<strObjID>
```

About Product Names in Siebel Configurator Scripts

Several scripting methods have product name as an argument. Product name in this context means the name of a component product you have added from the product table to a customizable product. You must specify product names in a way that makes them unique. You do this by specifying the product name, vendor name, vendor location, and attribute values.

Specify product names using the following syntax:

{ProductName; VendorName; VendorLocation}; AttributeName1=Value1; AttributeName2=Value2; ...

Observe the following guidelines when specifying product names:

- ProductName is required. All other arguments are optional.
- The order of items in the name is important. ProductName must be followed by VendorName. VendorName must be followed by VendorLocation. You cannot specify ProductName followed by VendorLocation.
- If ProductName is unique, you do not have to include VendorName or VendorLocation, and you do not need to enclose ProductName in braces.

Observe the following guideline when specifying the product path: do not use the sequence *#1* inside the product path. This sequence is invalid. For example, the following product path does not work: `$.[EXH SLCT-#1101 10X10 INLINE]#1.[CARPET, SPECIAL CUT (EX SEL10x10)]#[CARPET, SPECIAL CUT (EX SEL10x10)]`

The following are examples of product names:

- {ProductName; VendorName}; AttributeName=Value
- ProductName; AttributeName=Value
- ProductName

About Product Path in Siebel Configurator Scripts

The product path is the path from the root of a product with components to a component product within it. The path is a string that specifies the product with components root and all relationship names leading to the component product. All or part of the product path are arguments to several scripting methods.

The syntax for product paths is as follows:

$.[Root Product]#1.[Relationship]#[Component Product]

Observe the following guidelines for product paths:

- The $ before .[Root Product] refers to a special configuration object called the basket. The basket contains all the objects in the product with components.
- The #1 after [Root Product] refers to the first instance of the root product in the basket.
- Use a dot (.) to specify a relationship.
- Use a # to specify a component product within a relationship.
- Use .xa[Attribute Name] to specify the attribute name
- Use .xf[Value] to return the attribute value

- Use .xf[Quantity] to return the product quantity
- Use $Int.[<id>] to return the product's integration id. This is useful when multiple instances of the product exist. If it is not used and multiple instances of the product exist, the script selects at random which instance of the product it returns.
- Enclose relationship names and component product names in square brackets ([ ]). Use product name syntax to specify the name of a component product.
- All paths must end with a product name.

Figure 14 shows the structure of product with components CP1.

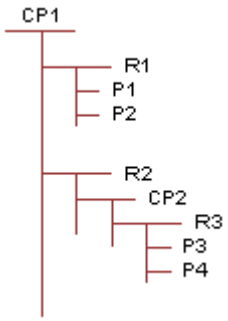

Figure 14. Product with Components Structure

Product with components CP1 has two relationships R1 and R2. Relationship R1 contains two component products P1 and P2. Relationship R2 contains the product with components CP2. CP2 contains one relationship R3, which has two component products P3 and P4.

The product paths for this product with components are as follows:

- **For P1.** $.[CP1]#1.[R1]#[P1]
- **For P2.** $.[CP1]#1.[R1]#[P2]
- **For CP2.** $.[CP1]#1.[R2]#[CP2]
- **For P3.** $.[CP1]#1.[R2]#[CP2].[R3]#[P3]
- **For P4.** $.[CP1]#1.[R2]#[CP2].[R3]#[P4]

Here are examples of using product paths in script methods:

1. To add 1 P1:

   ```
   AddItem(“$.[CP1]#1”, “R1”, “P1”, “1”)
   ```

2. To add 1 P3:

   ```
   AddItem(“$.[CP1]#1.[R2]#[CP2]”, “R3”, “P3”, “1”)
   ```

3. To reduce the quantity of P3 to 0:

   ```
   RemoveItem(“$.[CP1]#1.[R2]#[CP2].[R3]#[P3]”)
   ```

4 To set the attribute Color to Red for P1:

```
SetAttribute("$.[CP1]#1.[R1]#[P1]", "Color", "Red")
```

Siebel Configurator Script Events and Methods

Siebel Configurator constraint scripts use the following events and methods:

Cfg_InstInitialize Event

This event is called once for each session after the customizable product is instantiated and before any user requests are accepted. The customizable product selection pages do not display until all scripts associated with this event have finished.

Syntax

Cfg_InstInitialize (*RootProduct* as String)

Argument	Description
RootProduct	String. The name of the customizable product.

Returns

None.

Usage

Use this event to store global variables that will be reused, such as BusObjects and BusComps. This event is also useful for reading information from external sources such as forms or SmartScript.

The Cfg_InstInitialize event is always triggered for Root Customizable Products. However, it can also be triggered for child customizable products under the scenarios described here.

For example, you have the following product structure.

Root Customizable Product

|_Relationship

|_Child Customizable Product

When the default cardinality on the relationship is greater than 0 (that is, a child customizable product is initialized as part of the root customizable product) and the user customizes the root customizable product, the following events are triggered:

- For the root customizable product: Cfg_InstInitialize - Root CP
- For the child customizable product: Cfg_InstInitialize - Root CP

  NOTE: Cfg_InstInitialize - Root CP means that this event is triggered when it is defined with the root product = Root CP.

When the default cardinality on the Relationship is set to 0 (that is, a child customizable product is not initialized as part of the root customizable product) and the user customizes the root customizable product, the following event is triggered:

For the root customizable product: Cfg_InstInitialize - Root CP

If you try to add the child customizable product manually after the configuration session is loaded then Cfg_InstInitialize - Root CP is not triggered for the child customizable product.

Cfg_ChildItemChanged Event

After a user request is processed and the Siebel Configurator engine computes a new solution, this event is called for the product root. The event returns a property set containing all the products whose quantities have changed.

This event is also called if the user changes an item's quantity and the Request Conflict dialog box displays:

- If the user selects OK in the dialog box, this submits a request that reverses the last request. Since this revises the item's baseline quantity, the event is called for the item.
- If the user selects Cancel, previous conflicting requests are removed from the session. Since the current baseline values do not require revision, the event is not called for the item.

This event does not return items for which only attribute values have changed. For example, if the total number of 100 GB disk drives in a solution changes, this event returns a property set containing the 100 GB disk drive because the item quantity changed.

If the user enters or selects 10 feet as the desired value of the length attribute for power supply wiring, the returned property set does not contain power supply wiring since this is an attribute change.

In the selection pages for a customizable product, this event is called when the user selects an item. It is also called when the user increases or decreases the quantity of an item. This event is called before the Cfg_ItemChanged event.

Syntax

Cfg_ChildItemChanged (*ChangedItem* as Property Set)

The *ChangedItem* argument is passed as type PropertySet. This is a named XML element:

```
<ChangedItem ObjName= “objname” OldQty= “oldqty”
NewQty= “newqty”/>
```

The properties of this XML element are defined in the following table:

Property	Description
ObjName	String. The item name.
OldQty	String. The item quantity prior to the request.
NewQty	String. The new baseline item quantity.

Several Siebel API-related methods are needed to read data from the property set:

- **GetChildCount().** Returns the total number of changed items in the property set. Use this method to set the counter for a while-loop that reads the property set.
- **GetChild(*n*).** Returns the *nth* record in the property set. Use this method within a while-loop to read records from the property set.
- **GetProperty(“*argument*”).** Returns the value of *argument.* Allowed arguments are ObjName, OldQty, and NewQty. Arguments must be in quotes. Use this method to read the property values from each record in the property set.

Returns

None

Usage

Use this event to determine what changes have been made to products in the solution and to submit additional requests as needed. For example, you could track the memory requirements for software the user selects and submit requests to add the correct amount of RAM to a computer configuration.

When you submit a request that changes item quantities, the submission causes the event to be called and the script runs again. Be sure to insert logic in the script that prevents an infinite loop of request submissions.

Example

The following Siebel Visual Basic example writes to a file the item name, the old quantity, and the new quantity of all the items whose quantities change in each solution.

```
Sub Cfg_ChildItemChanged (ChangedItem As PropertySet)
   dim psItem as PropertySet
   dim n as Integer
   dim nCnt as Integer
   dim sObjName as String
   dim sOldQty as String
   dim sNewQty as String
   dim sMsg as String
   dim hndl as Long

   hndl = Freefile
   REM use a relative path to open cfgtest.log
   Open "..\cfgtest.log" for append as #hndl
   nCnt = ChangedItem.GetChildCount()
   For n = 0 to (nCnt -1)
      set psItem = ChangedItem.GetChild(n)
      With psItem
         sObjName = .GetProperty("ObjName")
         sOldQty = .GetProperty("OldQty")
         sNewQty = .GetProperty("NewQty")
      End With
      sMsg = "ObjName = " & sObjName
      sMsg = sMsg & "; OldQty = " & sOldQty
      sMsg = sMsg & "; NewQty = " & sNewQty
      Write #hndl, sMsg
      set psItem = Nothing
   Next
   Close #hndl
End Sub
```

Cfg_AttributeChanged Event

After a user request is processed and the Siebel Configurator engine computes a new solution, this event is called for the product root. The event returns a property set containing all the products whose attributes have changed.

This event is also called if the user changes an item's attribute and the Request Conflict dialog box displays:

- If the user selects OK in the dialog box, this submits a request that reverses the last request. Since this revises the item's baseline attribute value, the event is called for the item.

- If the user selects Cancel, previous conflicting requests are removed from the session. Since the current baseline values do not require revision, the event is not called for the item.

In the selection pages for a customizable product, this event is called when the user enters or changes an attribute value.

Syntax

Cfg_AttributeChanged (*ChangedAttribute* as Property Set)

The *ChangedAttribute* argument is passed as type PropertySet. This is a named XML element:

```
<Id ObjName =“objname”>

<AttName = “attribute name” OldVal= “old value”
NewVal= “newvalue”>

...

</Id>
```

The properties of this XML element are defined in the following table:

Property	Description
ObjName	String. The item name.
AttName	String. The attribute name.
OldVal	String. The attribute value prior to the request.
NewVal	String. The new baseline attribute value.
Id	String. The object ID of the item whose attribute value has changed.

Several Siebel API-related methods are needed to read data from the property set:

- **GetChildCount().** Returns the total number of changed items in the property set. Use this method to set the counter for a while-loop that reads the property set.
- **GetChild(*n*).** Returns the *nth* record in the property set. Use this method within a while-loop to read records from the property set.
- **GetProperty(“*argument*”).** Returns the value of *argument.* Allowed arguments are ObjName, OldQty, and NewQty. Arguments must be in quotes. Use this method to read the property values from each record in the property set.
- **GetType().** Retrieves the object ID of the item for which the attribute was changed.

Returns

None

Usage

Use this event to determine what changes have been made to product attributes in the solution and to submit additional requests as needed. For example, you could track the attributes selected for a product and submit requests based on them.

Example

The following example, writes to a file the item name, the old attribute value, and the new attribute value of all the items whose attribute values change in each solution.

```
{
var item;
var log = Clib.fopen("c:\\attchgd.log", "a");

   var id = ChangedAttribute.GetType();
   Clib.fputs(id, log);

   var nCnt = ChangedAttribute.GetChildCount();
   Clib.fputs(nCnt, log);

   for ( var i = 0; i<nCnt; i++ )

   {
      item = ChangedAttribute.GetChild(i);
      var attName = item.GetType();
      var oldV = item.GetProperty("OldVal");
      var newV = item.GetProperty("NewVal");
      var s = "AttName = " + attName;
      s = s + "; OldVal = ";
      s = s + oldV;
      s = s + "; NewVal = ";
      s = s + newV;
      Clib.fputs(s, log);
    }
    Clib.fclose(log);
}
```

Cfg_InstPostSynchronize Event

This event is called for the product root after the user clicks Done in the selection pages to end a configuration session. No further processing by the Siebel Configurator engine occurs in connection with this event. Using this event to adjust item quantities or attribute values is not recommended.

NOTE: This event is not supported for asset-based ordering. For asset-based ordering the Product Manipulation Toolkit Business Service (SIS OM PMT Service) handles the Synchronize call. For asset-based ordering, use the PMT service methods and the asset-based ordering workflows to write scripts.

Syntax

Cfg_InstPostSynchronize (*RootProd* as String)

Argument	Description
RootProd	String. The name of the customizable product.

Returns

None

Usage

Use this event to add or modify line item information before it is stored. You can also use this event to modify pricing or do other activities associated with quote line items.

Cfg_ItemChanged Event

After a user request is processed and the Siebel Configurator engine computes a new solution, this event is called for each component customizable product whose quantity has changed. The script associated with this event must be associated with the component customizable product.

For example, CP1 is a customizable product. One of its component products is customizable product CP2. A script inserted in the Cfg_ItemChanged event in CP2 runs when the quantity of CP2 changes while CP1 is being configured.

To set up a Cfg_ItemChanged script for the component customizable product CP2 you must do the following:

- Select CP2 in the product table and lock its work space.
- Open the script editor and select the Cfg_ItemChanged event.
- Specify CP1 as the root product for the script.
- Write the script, check its syntax, and save the script.
- Release a new version of CP2.

This event provides a simple way to write scripts for a customizable product that run only when that product is a component of another customizable product. This event is called after the Cfg_ChildItemChanged event.

Upgrade users: Use the Cfg_ChildItemChanged event to obtain functionality similar to the Cfg_ItemChanged event in release 6.x.

In the selection pages for a customizable product, this event is called when the user selects the component customizable product. It is also called when the user increases or decreases the quantity of that product.

Syntax

Cfg_ItemChanged (*ProdName, OldQty, NewQty*)

Argument	Description
ProdName	String. The name of the component customizable product. Use product name syntax
OldQty	Integer. The component customizable quantity prior to the request.
NewQty	Integer. The new baseline quantity for the component customizable product.

Items

Use this event only to write scripts in customizable products that will be components of other customizable products.

Returns

Returns the component customizable product name, the quantity of it in the solution prior to the user request, and the quantity after the user request. This event does not return changes in the quantity of the products comprising a component customizable product.

Cfg_OnConflict Event

This event is called for the product root when the Siebel Configurator engine encounters a conflict while computing a solution. A conflict is when a user action violates a constraint. The constraint can be in the declarative portion of the product or can be a user pick.

When this event is called, if no script is defined, the application's normal conflict resolution messages display. Typically, the user must add or remove items to resolve the conflict. Users can retain the last user-pick and undo a previous user-pick, or they can undo the last user pick.

If a script is defined, you can resolve the conflict in the script. The application does not prompt the user with a conflict message. Submitting a request in a script on this event is not recommended.

Syntax

Cfg_OnConflict (*Explanation, Resolution*)

Argument	Description
Explanation	String. Passes in the application message explaining the conflict
Resolution	String. Accepts as output one of the following values: ■ **UndoLastRequest.** This argument causes an undo of the last user request. ■ **RemoveFailedRequests.** This argument causes an undo on all user requests that cause the last request to fail. The last request is retained. If neither of the previous arguments is passed as output, the application presents the user with the normal conflict messages.

Returns

Returns the application error message and accepts one of two values as output. The outputs are forwarded to the Siebel Configurator engine and are used to resolve the conflict.

Usage

Use this event to manage conflict resolution in the background. For example, you can could create If-then statements that pass one of the outputs depending on a configuration condition in the model. If the conditions for handling the output do not exist, then passing no value causes the normal conflict resolution message.

OnAttributeSelected Event

This event is triggered when an attribute has been selected and the model update is about to occur.

Syntax

Cfg_AttributeSelected (SelectedAttribute)

Returns

None

OnChildItemSelected Event

This event is triggered before the Item is added to the Instance.

Syntax

Cfg_ChildItemSelected (SelectedItem)

Returns
None

GetInstanceId Method

This method returns the row ID of the product root in the source object, such as the quote or order line item. The row ID of the product root will be different for each quote or order.

Syntax
GetInstanceId() as String

Argument	Description
None	

Items
Not applicable.

Returns
Returns the row ID of the customizable product root.

GetCPInstance Method

This method returns the entire structure of a customizable product as a property set.

Syntax
GetCPInstance(ps)

Argument	Description
ps	Can be created with TheApplication().NewPropertySet();

Items
Not applicable.

Returns
Returns the structure of the customizable product as a property set. Depending on the structure of the customizable product, the property set can be complex. To learn how to access the property set, use the following JavaScript code to dump the property set to a file. You can then study the property set structure to determine how to access it using a script.

```
/*
Use the PropertySetToFile(PropSet, fileName, title) API in your script.
   Note that fileName must be double slashed, as demonstrated in the example below:
      PropertySetToFile(InputsPS, "..\\temp\\testPSexport.txt", Inputs into “ +
MethodName);
   This will write the property set to a text file in the Siebel temp directory.
*/
function PropertySetToFile (PropSet, fileName, title)
{
   var file = Clib.fopen(fileName, "at");
   LogData(("\n-------------------------------------------------"), file);
   LogData(("Start Process  " + Clib.asctime(Clib.gmtime(Clib.time()))), file);
   LogData(title, file);
   LogData("PROVIDED PROPERTY SET", file);
   WritePropertySet(PropSet, file, 0);
   Clib.fclose(file);
   return (CancelOperation);
}
function WritePropertySet(PropSet, file, Level)
{
   if ((Level == "") || (typeof(Level) == "undefined")){
         Level = 0;
   }
   var indent = "";
   for (var x = 0; x < Level; x++){
         indent += "\t";
   }
   var psType = PropSet.GetType();
   var psValue = PropSet.GetValue();
   LogData((indent + "Type: " + psType + " Value: " + psValue), file);
```

```
   var propName = PropSet.GetFirstProperty();
   while (propName != ""){
      var propValue = PropSet.GetProperty(propName);
      LogData((indent + propName + " = " + propValue), file);
      propName = PropSet.GetNextProperty();
   }
   var children = PropSet.GetChildCount();
   for (var x = 0; x < children; x++){
         LogData(( indent + "CHILD PROPERTY SET " + x), file);
         WritePropertySet(PropSet.GetChild(x), file, (Level + 1));
   }
}
function LogData(DataString, file)
{
   try {
      Clib.fputs((DataString + "\n"), file);
      Clib.fflush(file);
   }
   catch (e){
         // no action
   }
}
```

GetObjQuantity Method

This method returns the quantity of the specified component product in the current solution.

Syntax

GetObjQuantity (*ProdName*) as Integer

Argument	Description
ProdName	String. The name of the component product. Use product name syntax to identity the component product.

Items

Can be used only for component products within the customizable product.

Returns

Quantity of the component product in the current solution as an integer. If multiple instances of the component exist, the quantity of all instances are added together and the sum is returned. For example if there are two instances of an item one with quantity five and the other with quantity three, the return is eight.

Example

This script fragment obtains the quantity of 10 GB Drive (vendor = Sony) in the current solution and assigns it to the variable iItemQty.

```
Dim iItemQty as Integer

iItemQty = GetObjQuantity(“{10 GB Drive; Sony}”)
```

AddItem Method

This method creates a new instance of an item and adds the specified quantity to the solution. For example, Item A exists in the solution and has quantity three. You use AddItem to add two more Item A to the solution. The new solution will contain two instances of Item A, one with quantity three, and one with quantity two.

Syntax

AddItem (*ParentObjPath, RelName, ProdName, Quantity*) as Integer

Argument	Description
ParentObjPath	String. The product path to, but not including, the relationship in which the component product resides. Use product path syntax to specify the path. If the component product is located at the product root, then specify the product root as the *ParentObjPath.*
RelName	String. The name of the relationship containing the component product you want to add. If the component product is located at the product root, then specify the customizable product name as the *RelName*.
ProdName	String. The name of the product you want to add. Use product name syntax to specify the product name.
Quantity	String. The amount of the product you want to add.

Items

Can be used only for component products within the customizable product.

Returns

Returns the integration id of the item added if the add was successful. Returns 0 if the add fails.

Example

For an example of using AddItem, see "About Product Path in Siebel Configurator Scripts" on page 305.

RemoveItem Method

This method reduces the quantity of the specified item to 0 in the current solution. If multiple instances of an item have the same path, the Siebel Configurator engine randomly picks one of the instances and removes it.

Syntax

RemoveItem (*ObjPath*) as Integer

Argument	Description
ObjPath	String. The full path of the component product you want to remove. Use product path syntax to specify the path.

Items

Can be used only for component products within the customizable product.

Returns

Returns 1 if the item removal was successful. Returns 0 if the removal fails.

Example

For an example of using RemoveItem, see "About Product Path in Siebel Configurator Scripts" on page 305.

SetAttribute Method

This method sets the value of an attribute for an item in the customizable product. This method can also be used to set attribute values for attributes of the customizable product.

If multiple instances of an item have the same path, the Siebel Configurator engine randomly picks one instance and changes its attribute values. Users can reference a particular instance of a product by using $Int.[<id>] to specify the integration id in the product path.**Syntax**

SetAttribute (*ObjPath, AttName, AttVal*) as Integer

Argument	Description
ObjPath	String. The full path of the component product. For attributes of the customizable product, specify the product root. Use product path syntax to specify the path.
AttName	String. The name of an attribute of the component product or customizable product.
AttValue	String. The value to which you want to set the attribute.
LICflag	Optional input. String. This property is applicable for LOV Domains. It specifies whether the value passed to the method is the LOV display value or the LOV language independent code (LIC). Set it to 'Y' if you are supplying the LIC value.

For LOV domains, the *AttValue* must be one of the values in the list of values. Validation expressions defined for LOV domains are ignored.

For range of value domains, the *AttValue* must be within the domain defined by the validation expression.

Items

Can be used only for component products within the customizable product.

Returns

Returns 1 if setting the attribute was successful. Returns 0 if setting the attribute failed.

Example

For an example of using SetAttribute, see "About Product Path in Siebel Configurator Scripts" on page 305.

GetAttribute Method

The GetAttribute method gets the value of an attribute for an item in the customizable product. This method can also be used to get attribute values for attributes of the customizable product.

If multiple instances of an item have the same path, the Siebel Configurator engine randomly picks one instance. Users can reference a particular instance of a product by using $Int.[<id>] to specify the integration id in the product path.

Syntax

GetAttribute (ObjPath, AttName) as String

Input Arguments

Argument	Description
ObjPath	The full path of the component product. For attributes of the customizable product, specify the path of the root product. Use product path syntax to specify the path.
AttName	The name of an attribute of the component product or customizable product.
Items	The name of component products. Can be used only for component products within the customizable product.

Output

This API returns a string. If it is successful, the string contains the Language Independent Code (LIC) value of the attribute. If it is not successful, the string is empty.

Examples

```
Ptype =  GetAttribute("$.[Life Insurance]#1", "PlanType")

AttVal =  GetAttribute("$.[RootProd1]#1.[Rel1]#[ChildProd1]", "attr1");

AttVal = GetAttribute ("$Int.[12-4JEFK]", "Color");
```

BatchRequest Method

This method is used to stack multiple configuration requests into a single request, improves performance because it is not necessary to initialize the constraint engine for each request.

For example, when a user exits Siebel Configurator, you want to automatically add items A, B, remove item C and set attribute values for newly added items A and B. Without the BatchRequest method, the constraint engine is called for each configuration request such as AddItem, RemoveItem and SetAttribute. With the BatchRequest method, you can write a script that makes a single call to the constraint engine to compute the solution, which improves performance.

Syntax

BatchRequest (InputArgs, OutputArgs)

Argument	Description
InputArgs	A collection of property sets. Each child property set contains a Siebel Configurator script request along with its required set of parameters. Optionally, each child property set can contain alias properties that may be used as a replacement for product paths. There are two type of alias properties: ■ **OutItemAlias.** User can give any name value as an Alias. Applies only to "AddItem" script requests. ■ **ItemAlias.** This can be any one of the given values for OutItemAlias. Optionally, any child request can use the ItemAlias property instead of the product path. The user can use any name as the value of the alias.
OutputArgs	A property set that contains the Integration IDs returned by the AddItem script request and any errors that might have occurred during this BatchRequest execution.

Example

This example shows one way that you can use batch request. It assumes the product is set up as follows:

```
T-Shirt

|_Team T-Shirt (relationship)

   |_Short sleeve shirt (component product) with Attribute: Color (dropdown values:
Please Specify, Red, Blue)

   |    |_Accessories (relationship)

   |       |_Matching Hat (component product)

   |        |_Matching Bag (component product) with Attribute: Type (dropdown value:
Please Specify, Duffel, Tote)

    |_Long Sleeve Shirt (component product)
```

The following script adds 10 red short sleeve shirts along with 10 matching hats and 15 blue short sleeve shirts along with one matching duffel bag

```
function Cfg_InstInitialize (RootProduct)

{
```

```
var parentPropset = TheApplication().NewPropertySet();
var outPropset = TheApplication().NewPropertySet();

var childPropset1 = TheApplication().NewPropertySet();
var childPropset2 = TheApplication().NewPropertySet();
var childPropset3 = TheApplication().NewPropertySet();
var childPropset4 = TheApplication().NewPropertySet();
var childPropset5 = TheApplication().NewPropertySet();
var childPropset6 = TheApplication().NewPropertySet();
var childPropset7 = TheApplication().NewPropertySet();

//Add 10 short sleeve shirts
childPropset1.SetProperty("RequestType", "AddItem");
childPropset1.SetProperty("OutItemAlias","RedShirtAlias");
childPropset1.SetProperty("Parent Path","$.[T-Shirt]#1");
childPropset1.SetProperty("PortName","Team T-Shirt");
childPropset1.SetProperty("Name","Short sleeve shirt");
childPropset1.SetProperty("Quantity","10");
parentPropset.AddChild(childPropset1);

//set Red color
childPropset2.SetProperty("RequestType","SetAttribute");
childPropset2.SetProperty("ItemAlias","RedShirtAlias"); // picks particular instance
//childPropset2.SetProperty("Path","$.[T-Shirt]#1.[Team T-Shirt]#[Short sleeve
shirt]"); // picks random instance
childPropset2.SetProperty("Name","Color");
childPropset2.SetProperty("Value","Red");
parentPropset.AddChild(childPropset2);
```

```
//Add 10 Matching Hats

childPropset3.SetProperty("RequestType", "AddItem");

childPropset3.SetProperty("ItemAlias","RedShirtAlias"); // picks particular instance

//childPropset3.SetProperty("Parent Path","$.[T-Shirt]#1.[Team T-Shirt]#[Short sleeve
shirt] "); // picks random instance

childPropset3.SetProperty("PortName","Accessories");

childPropset3.SetProperty("Name","Matching Hat");

childPropset3.SetProperty("Quantity","10");

parentPropset.AddChild(childPropset3);


//Add 15 short sleeve shirts

childPropset4.SetProperty("RequestType", "AddItem");

childPropset4.SetProperty("OutItemAlias","BlueShirtAlias");

childPropset4.SetProperty("Parent Path","$.[T-Shirt]#1");

childPropset4.SetProperty("PortName","Team T-Shirt");

childPropset4.SetProperty("Name","Short sleeve shirt");

childPropset4.SetProperty("Quantity","15");

parentPropset.AddChild(childPropset4);


//set Blue color

childPropset5.SetProperty("RequestType","SetAttribute");

childPropset5.SetProperty("ItemAlias","BlueShirtAlias"); // picks particular instance

//childPropset5.SetProperty("Path","$.[T-Shirt]#1.[Team T-Shirt]#[Short sleeve
shirt]"); // picks random instance

childPropset5.SetProperty("Name","Color");

childPropset5.SetProperty("Value","Blue");

parentPropset.AddChild(childPropset5);


//Add 1 Matching Bag

childPropset6.SetProperty("RequestType", "AddItem");

childPropset6.SetProperty("ItemAlias","BlueShirtAlias"); // picks particular instance
```

```
//childPropset6.SetProperty("Parent Path","$.[T-Shirt]#1.[Team T-Shirt]#[Short sleeve
shirt] "); // picks random instance

childPropset6.SetProperty("OutItemAlias","BlueDuffelBag");

childPropset6.SetProperty("PortName","Accessories");

childPropset6.SetProperty("Name","Matching Bag");

childPropset6.SetProperty("Quantity","1");

parentPropset.AddChild(childPropset6);


//Set Bag Type = Duffel

childPropset7.SetProperty("RequestType","SetAttribute");

childPropset7.SetProperty("ItemAlias","BlueDuffelBag"); // picks particular instance

//childPropset7.SetProperty("Path","$.[T-Shirt]#1.[Team T-Shirt]#[Short sleeve
shirt].[Accessories]#[Matching Bag]")  // picks random instance

childPropset7.SetProperty("Name","Type");

childPropset7.SetProperty("Value","Duffel");

parentPropset.AddChild(childPropset7);


BatchRequest(parentPropset, outPropset) ;


}
```

Creating Siebel Configurator Event Scripts

Event scripts run when defined events are called during a configuration session. You create an event script by selecting an event method and writing the script within it.

Event scripts can call methods and objects defined in the Siebel API. They can also call methods assigned to the declarations area.

To create an event script

1. Navigate to the Administration - Product screen.
2. Select and lock the desired product or product class.
3. Navigate to the Scripts view for the Work Space version.

4 In the Scripts list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Select the desired event. If this is not the first script for this event, overwrite the event name with a script name. All scripts for a customizable product must have a unique script name.
Program Language	If this is the first script you are creating for this product, select Visual Basic or eScript. If this is not the first script, and select the same programming language used for previous scripts.
Root Product	Select the current customizable product.

5 Enter the script in the area in the Script Definition form.

The Script Definition form is located after the Scripts list.

6 When you have finished entering the script, click Check Syntax.

If there are errors, they will display at the start of the Customizable Product Scripts form. Correct any errors before saving the script.

7 In the Script Definition form, click Save.

Creating Siebel Configurator Declarations Scripts

Declaration scripts are methods that you want to make available to event scripts or other declaration scripts. Declaration scripts are stored in a common area accessible by event scripts. Declaration scripts can call methods and objects defined in the Siebel API.

Use declaration scripts to write methods that are common to more than one event script. Instead of repeating the method in each event script, you can write one declaration script and call it from within the event scripts that use it.

To create a declarations script

1 Navigate to the Administration - Product screen.

2 Select and lock the desired product or product class.

3 Navigate to the Scripts view for the Work Space version.

4 In the Scripts list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Name	Choose “(declarations).” If this is not the first declarations script, overwrite “(declarations)” with a script name. All scripts for a customizable product must have a unique script name.
Program Language	If this is the first script you are creating for this product, select Visual Basic or eScript. If this is not the first script, and select the same programming language used for previous scripts.
Root Product	Select the current customizable product.

5 Enter the script in the area in the Script Definition form.

The Script Definition form is located after the Scripts list.

6 When you have finished entering the script, click Check Syntax.

If there are errors, they will display at the start of the Customizable Product Scripts form. Correct any errors before saving the script.

7 In the Script Definition form, click Save.

8 From the Scripts view menu, select Validate.

This starts a configuration session. Verify that the new script works correctly.

Reviewing the Siebel Configurator Script Log File

If a method within a script fails or the script contains syntax errors, you can obtain valuable diagnostic information by reviewing the script log.

To review the script log file

1 Navigate to `\log\cfgscript.log`. It is located in the Siebel installation directory.

2 Open the file with a text editor.

3 Locate the entries for the date and time the script ran.

About Managing Siebel Configurator Scripts

You can manage scripts in the following ways:

- "Editing Siebel Configurator Scripts" on page 329
- "Deleting Siebel Configurator Scripts" on page 329

Editing Siebel Configurator Scripts

You can edit scripts by selecting the script in the Scripts view. If you edit declaration scripts, verify that the changes do not adversely affect event scripts.

To edit a script

1. Navigate to the Administration - Product screen.
2. Select and lock the desired product or product class.
3. Navigate to the Scripts view for the Work Space version.
4. Highlight the script you want to edit.

   The script displays in Script Definition form, located after the list of scripts.
5. In the Script Definition form, edit the script.
6. When you have finished editing the script, click Check Syntax.

   If there are errors, they will display at the start of the Customizable Product Scripts form. Correct any errors before saving the script.
7. In the Script Definition form, click Save.

Deleting Siebel Configurator Scripts

After deleting a script, test the customizable product in validation mode to verify that the product works correctly.

To delete a script

1. Navigate to the Administration - Product screen.
2. Select and lock the desired product or product class.
3. Navigate to the Scripts view for the Work Space version.
4. In the Scripts list, select the script you want to delete.
5. In the Scripts list, click Delete.

6 Open the Scripts view list menu and click Validate.

This starts a configuration session. Verify that the customizable product functions correctly.

20 Testing Products and Using Workspace Projects

This chapter describes the two ways to test products:

- **Validation Mode.** Allows you to test the product by working with the same Siebel Configurator user interface that end users would use. This applies to products with components.
- **Scenario Tester.** Allows you to test multiple scenarios for the product, which you create in Workspace Projects view, and it returns the results in an XML file. This applies to products with components and products with attributes.

This chapter includes the following topics:

Testing a Product with Components in Validation Mode

You can test a product with components in validation mode. This creates an instance of the product with components and presents its selection pages. You can test configuration constraints, the user interface, and pricing exactly as if you were a user.

Validation mode can create an instance of the product with components based on the work space or on any released version of the product

Validation mode uses a specific quote with following name convention:

```
Customizable Product Validation Quote_[username]
```

For example, if the username is SADMIN, the quote must be named Customizable Product Validation Quote_SADMIN.

NOTE: If you make any fields required in the Quote business component, create the quote manually for each user (using the naming convention described here) and populate the required fields on each Quote before validating customizable products for the first time.

Test the product with components in the following ways:

- To test a specific group of configuration rules, set all the rules you do not want to test to inactive and then go to validation mode to test the desired rules. Then activate configuration constraints one at a time as needed and return to validation mode to test the result.
- If you have purchased Siebel Pricer, be sure to fully validate all component-based pricing adjustments. If you are using automatic pricing updates, verify that selection pages redisplay fast enough after each user action. If redisplay is too slow, consider switching to a base theme for the user interface that uses manual price updates. When in validation mode, the application uses the price list assigned to a special quote called Customizable Product Validation Quote_[username]. You can test the product with multiple price lists by assigning each of the price lists to this special quote.
- Verify that user access is set up correctly for all the components of the product with components. Do this by checking the categories to which the product with components and all its components are assigned. Then check the access control groups assigned to these categories and associated catalogs. Users who will configure the product must have access permission to the product and all its components. You can check category assignments in Product Administration, then the Category or in Catalog Administration.

Validate a product with components at regular intervals while you are developing it. For example, after you enter a block of related configuration rules or after customizing the selection pages, go to validation mode and check your work.

If your products with components are complex, consider developing written test plans that exercise all the configuration rules and all expected user behaviors. In particular, be sure to test for unexpected or incorrect user behaviors in order to rule out unexpected responses from the Siebel Configurator engine.

You enter validation mode by clicking the Validate button, which located in the views where you work with products with components, such as the Product Versions view, and the Constraints view.

To test a product with components in validation mode

1. Make changes to the product with components.

   For example, add configuration rules in the Constraints view.
2. Navigate to the Administration - Product screen, then the Product Definitions view.
3. In the Products list, select the desired product with components.
4. In the Versions list, select either the Work Space record or the record for a released version of the product.
5. In the Versions list, click Validate.

   The application creates an instance of the product and displays it in Siebel Configurator.

Testing Product with Components Pricing

If you want to test pricing, you must associate a price list with the Customizable Product Validation Quote_[username] quote. This is a special quote provided for validating products with components.

To test product with components pricing

1. Navigate to the Quotes screen.
2. In the Quotes list, query for the Customizable Product Validation Quote_[username] quote.
3. Assign the desired price list to the Customizable Product Validation Quote_[username] quote.
4. Navigate to the Administration - Product screen, then the Product Definitions view.
5. In the Versions list, select either Work Space or a released version of the product.
6. Click Validate.

   The application creates an instance of the product and displays it in Siebel Configurator.

About Scenario Tester and Workspace Projects

Scenario Tester allows you to test the validity of your customizable product or simple product with attributes at any time that you specify (past, current, or future) and to create a number of quotes, orders, and agreements that will be used to test your customizable product. Each quote, order, agreement or asset is a scenario used for testing.

Before you test products using Scenario Tester, you must create a workspace project, which includes:

- **Contents list.** Add the objects you want to test.
- **Scenarios list.** Define the test Quotes, Sales Orders, Service Orders, and Agreements that you are using to test these objects.

Working with Versioned and Unversioned Objects

Scenario Tester can test:

- **Versioned objects.** You can test customizable products, classes, and attribute definitions.
  - If the objects are part of the active Workspace Project, Scenario Tester uses the version in the Workspace Project, even if it is unreleased.
  - If objects are not part of the Workspace Project, Scenario Tester uses the released version of the object.

    NOTE: Scenario Tester uses objects that are not part of the Workspace Project if objects that are part of the Workspace Project are dependent on them. For example, if you add a product with attributes to the Workspace Project, but do not add the Class that the product inherits its attributes, Scenario Tester uses the released version of that Class.
- **Unversioned objects.** Scenario Tester tests the eligibility, compatibility, and pricing rules for unversioned objects, using the Base Date to determine whether a particular item must be retrieved.

Administering Data in Test Mode

Administering data in test mode allows the administrator to create data that use either unreleased or future data. For example, in test mode, the administrator can enter a future price for an unreleased product or future-effective product.

You work in slightly different ways in these two modes:

- **Test mode.** The mode that the application is in when the Use Project checkbox is selected for a workspace project. This mode gives the administrator access to unreleased objects included in the workspace project and other objects not in the workspace project based on Base Date.
- **Normal mode.** The mode that the application is in when the Use Project checkbox is not selected for a workspace project. The application uses released objects based on the application's internal date.

When you are administering a Workspace Project in test mode, you can see unreleased products and tie them to other objects in the application for administration purposes, as follows:

- **Tying a Versioned Object to a Versioned Object.** You can associate a versioned object with another versioned object in test mode. An example is associating an attribute definition to a product class.
- **Tying a Versioned Object to a Unversioned Object.** You can view all unversioned objects regardless of test date and whether you are in Test Mode. For instance, you can view all price list items regardless of their effectivity dates and regardless of test date or mode. You can associate versioned objects with unversioned objects as follows:
  - **Price Lists, Cost Lists, Rate Lists, Aggregate Discounts, Discount Matrices, Eligibility and Compatibility Rules, Bundled Products, Product Recommendations.** In Test Mode, the application shows records if the item has been added to the Workspace Project, and shows records that are in effect on the test date if the item has not been added to the Workspace Project. In Normal Mode, the application shows records for products or classes that have released versions as of today.
  - **CP Pricing Designer.** In Test Mode, the application shows the component products of the version added to the Workspace Project, and shows the version of the product that is in effect on the Test Date if the item has not been added to the Workspace Project. In Normal Mode, the application shows the union of all component products across versions of a customizable product within the effective date period of the root customizable product price list item.
  - **Catalog Product Items.** In Test Mode, the application shows records if the item has been added to the Workspace Project, and shows records that are in effect on the test date if the item has not been added to the Workspace Project. In Normal Mode, the application shows records for products or classes that have released versions as of today. The record counter for products in a given category are based on today's date. While testing, you must run the counter update periodically on the production application.
  - **Product Promotions.** In Test Mode, the application allows the user to add products and classes contained in a Workspace Project as products and classes covered by the promotion. After testing, the user must ensure that the objects referred from the promotion are released during the effectivity dates of the promotion, so there are no integrity problems to run-times in Normal Mode.

- **Catalog Categories.** This object is used based on effective dates. It behaves the same way in Test Mode as in Normal Mode, because it does not have to associated with another object to be used.
- **Compound Product Rules, Simple Validation Rules, Cardinality Rules, Smart Part Numbers, Favorites, Attribute Adjustments, Data Validation Rules.** These objects are not supported by Scenario Tester.

Process of Testing Products with Scenario Tester

To use Scenario Tester Project to test products, perform the following tasks:

- "Defining a Workspace Project for Scenario Testing" on page 335
- "Defining the Contents for Scenario Testing" on page 336
- "Creating Scenarios for Scenario Testing" on page 337
- "Validating Scenarios" on page 339
- "Correcting Product Definitions and Retesting" on page 340

Defining a Workspace Project for Scenario Testing

The workspace project is a container that holds all the other information needed for scenario testing. It also is used to specify the date that will be tested.

This task is a step in "Process of Testing Products with Scenario Tester" on page 335.

To define a workspace project for the scenario

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the scenario.
Effective Start Date	Enter the date that is used to set the Start Date of the versions released from the Workspace Project. This value is also the date that the application uses to determine what versions must be exported for dependent items when the user chooses to export the full structure of the items in the Workspace Project.

Field	Comments
Base Date	Enter the date for the test. All the Quotes, Orders, and other scenarios that you create will be tested using the released versions, pricing, eligibility, compatibility, recommendations, promotions, and other specifications effective on this date. If this date is entered, it is used as the date for dependent items for full structure export. If both Effective Starting Date and Base Date are filled, Base Date is used, as long as the data is not in the workspace itself.
Use Project	Select this checkbox to enter test mode. For more information about test mode and normal mode, see "Administering Data in Test Mode" on page 334. You must select this checkbox to display test quotes and other test documents in the Scenarios view. If you uncheck it, these test documents will disappear from Scenarios view. When you select this check box, the application uses the items in the Workspace Project as if they are currently active. The Scenario Tester uses the workspaces of the items in the Workspace Project as the active versions, allowing the user to test them without releasing them. Selecting this check box makes the Workspace Project active only for the duration of the session for the current login. The user can select this check box to associate items in the Workspace project with one another without having to release them. If the user exports data, if the Use Projects flag is set, the objects in this workspace are exported. This field controls whether test documents are visible to the user. When it is selected, the test documents become visible. When not, the documents are not visible.

Defining the Contents for Scenario Testing

When you define the contents for the scenario, you specify all the products, attributes, classes, signals, and variable maps whose workspace versions will be tested. For these objects, the testing date from the joint workspace will not be in effect.

For example, this list can include one customizable product plus all the component products, product attributes, and product classes that are part of that customizable product. In more complex cases, it can include several customizable products and component products, product attributes, and product classes that are part of it.

This task is a step in "Process of Testing Products with Scenario Tester" on page 335.

To define the contents for the scenario test

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, select the workspace project for this scenario test.
3. Click the Contents view tab.
4. In the Contents list, click Add to add a new record.
5. In the Select Versioned Object dialog box, select an object.
6. Continue to add new records and to select new versioned objects until you have selected all the objects to be tested.

Creating Scenarios for Scenario Testing

Next, you create quotes, orders, and agreements. These scenarios are created as test documents, so that you can distinguish them from actual documents. These are the scenarios that will be run under the conditions that you specified for this scenario testing.

This task is a step in "Process of Testing Products with Scenario Tester" on page 335.

To create a quote for scenario testing

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, select the workspace for this scenario test.
3. Click the Scenarios view tab.
4. In the Scenarios view link bar, click Test Quotes.
5. Add a record to the Test Quotes list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the test quote
Account	Select an account to be used in the test quote.

6. Click the hyperlink in the Name field of the test quote.

   The standard Quotes screen appears.

7. Enter the details of the quote in the usual way.

   For more information about entering a quote see *Siebel Order Management Guide*.

8. Optionally, when you are creating the quote, click the price fields to display a pricing waterfall which shows how the final price was arrived at. Verify that this is the pricing behavior that you want.
9. Optionally, when you are creating the quote, click Customize to display the product in Siebel Configurator. Verify that this is the interface that you want.

To create a service order for scenario testing

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, select the workspace for this scenario test.
3. Click the Scenarios view tab.
4. In the Scenarios view link bar, click Test Service Orders.
5. Add a record to the Test Service Orders list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Order #	Displays an application generated number used to identify the service order.
Account	Select an account to be used in the test quote.

6. Click the hyperlink in the Order # field of the test service order.

   The standard Service Orders screen appears.

7. Enter the details of the service order in the usual way.

   For more information about entering an order see *Siebel Order Management Guide*.

To create a sales order for scenario testing

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, select the workspace for this scenario test.
3. Click the Scenarios view tab.
4. In the Scenarios view link bar, click Test Sales Orders.
5. Add a record to the Test Service Orders list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Order #	Displays an application generated number used to identify the service order.
Account	Select an account to be used in the test quote.
Price List	Select a price list to be used in the test quote.

6. Click the hyperlink in the Order # field of the test service order.

   The standard Sales Orders screen appears.

7. Enter the details of the sales order in the usual way.

   For more information about entering an order, see *Siebel Order Management Guide*.

To create an agreement for scenario testing

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, select the workspace for this scenario test.
3. Click the Scenarios view tab.
4. In the Scenarios view link bar, click Test Agreements.
5. Add a record to the Test Agreements list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the test agreement.
Account	Select an account to be used in the test agreement.

6. Click the hyperlink in the Name field of the test agreement.

   The standard Quote screen appears.

7. Enter the details of the agreement in the usual way.

   For more information about entering an agreement, see *Siebel Field Service Guide*.

Validating Scenarios

After setting up the scenarios, you can validate them.

NOTE: In addition to the verification described here, you can batch validate scenarios, as described in "Batch Validating Scenarios" on page 341.

This task is a step in "Process of Testing Products with Scenario Tester" on page 335.

To validate scenarios

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, select the workspace for this scenario test and select the Use Project check box.
3. Click the Scenarios view tab.
4. In the link bar of the Scenarios view, click either Test Quotes, Test Service Orders, Test Sales Orders, or Test Agreements.
5. In the Scenarios list, select one or more test quotes, test service orders, test sales orders, or test agreements.
6. Double click the test scenario and perform the usual tasks for that type of document.

   For example, if you are testing a quote, you must add products to the quote, look at the products, look at the prices, and so on.

Correcting Product Definitions and Retesting

If the scenario is invalid, you must correct the definitions of products and prices and retest the scenario.

For example, if one of the component products is not available on the date that this scenario is testing, you must modify your product list so the product is available on that date.

This task is a step in "Process of Testing Products with Scenario Tester" on page 335.

Displaying Only the Project in Use

If you have a long list of projects in the Workspace Projects list, it can be useful to display only the project you are currently working on and to hide the other projects.

To display only the project in use

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, in the record for the project you are currently using, select the Use Project checkbox.

   Select this checkbox in only one record.
3. Click Show Project in Use.

   Only the project you are currently using is displayed in the list.

To display all workspace projects again after displaying only the project in use

1. In Workspace Projects list, click Query.
2. Without entering any query criteria, click Go.

   All projects are displayed.

Working with the Scenario XML File

When you verify a scenario, then you can export that scenario as an XML file. The XML file contains the property set for that scenario, with all the properties that define the quote, order, asset, or agreement.

After the XML file has been saved, you can use any text editor to edit the property set in the XML file rather than changing the definition in the Quotes, Orders, or Agreements screen.

Then you can import the XML file into the application to update the scenario with the changes you made in the XML file.

To export a scenario to an XML file

1. Navigate to the Administration - Product screen, then the Workspace Projects view.

2 In the Workspace Projects list, select the workspace for this scenario test.

3 Click the Scenarios view tab.

4 In the link bar of the Scenarios view, click either Test Quotes, Test Service Orders, Test Sales Orders, or Test Agreements.

5 In the Scenarios list, select a test quotes, test service order, test sales order, or test agreement.

6 From the menu in the Scenarios view, select Manage Scenario.

7 In the Manage Scenario dialog box, click Export Scenario.

8 In the File Download dialog box, click Save.

9 In the Save As dialog box, specify the file name and directory and click OK.

To import a scenario from an XML file

1 Navigate to the Administration - Product screen, then the Workspace Projects view.

2 In the Workspace Projects list, select the workspace for this scenario test.

3 Click the Scenarios view tab.

4 In the link bar of the Scenarios view, click either Test Quotes, Test Service Orders, Test Sales Orders, or Test Agreements.

5 From the menu in the Scenarios view, select Manage Scenario.

6 In the Manage Scenario dialog box, click Browse, and use the Choose File dialog box to select the XML file you want to import.

7 In the Manage Scenario dialog box, click Import Scenario.

CAUTION: If there is a scenario with the same name and type, the scenario being imported from XML overwrites the existing scenario.

Batch Validating Scenarios

The topic "Validating Scenarios" on page 339 describes how to validate one or more scenarios in one of the scenario lists. Using this method, all the scenarios you validate must be either in Test Quotes, Test Service Orders, Test Sales Orders, or Test Agreements list.

You can also batch verify scenarios across these lists.

To batch validate scenarios

1 Navigate to the Administration - Product screen, then the Workspace Projects view.

2 In the Workspace Projects list, select the workspace for this scenario test.

3 Click the Scenarios view tab.

4 From the menu in the Scenarios view, select Manage Scenario.

5 In the Manage Scenario dialog box, click Batch Validate.

If there are errors, the application displays error messages in popup applets.

21 Releasing Products and Other Versioned Objects

This chapter discusses how to release new versions of products. For users who use separate development and production environments, it also covers how to migrate products between these two environments. This chapter includes the following topics:

About Versions of C/OM Objects

The information in this chapter applies to how versioning works for the following customer order management objects:

- **Customizable Products.** For general information about customizable products, see Chapter 6, "Products with Attributes" and Chapter 9, "Designing Products with Components."
- **Product Classes.** For general information about product classes, see Chapter 6, "Products with Attributes."
- **Attribute Definitions.** For general information about attribute definitions, see Chapter 6, "Products with Attributes."
- **Variable Maps.** For general information about variable maps, see *Siebel Order Management Infrastructure Guide*.
- **Signals.** For general information about signals, see *Siebel Order Management Infrastructure Guide*.

This chapter discusses how to work with versions of products with components, but the instructions also apply to the other objects, using the appropriate view.

This type of versioning allows you to create and administer multiple development and production versions of an item can be created and administered independently. It allows you to develop, test, and deploy items either in your production environment or in a separate development environment. It support a four-stage process for introducing new items: development, testing, approval, and production.

NOTE: Simple products are also versioned objects and must be released to make them visible to users, but they work in a way that is simpler than the versioned objects described in this chapter. For more information, see "Releasing a Simple Product" on page 34.

About Avoiding Duplicate Versioned Objects During Product Data Migration

Starting in Siebel 7.8, the import function for product data uses the object identifier to determine whether records being imported already exist in the target database. In earlier versions, the import function used a name-based identifier.

The object identifier refers to the value in the OBJECT_NUM column of the versioned object table, S_VOD, for the corresponding versioned object. Object identifiers are initially set as the value of the row ID. They are always persistent and are referenced in other tables, including S_PROD_INT. For this reason, it is now absolutely mandatory to preserve the object identifiers of versioned objects across multiple environments in order to consistently export and import these object definitions across different environments. This behavior affects all versioned objects: Products, Classes, Attributes, Signals, and Variable Maps.

Guidelines to Avoid Duplicate Versioned Object Records

Follow these guidelines to avoid creating duplicate versioned object records:

- Use a single master database for product administration. This database ensures that object identifiers of objects can be preserved and easily migrated to different environments.
- All other databases must have the same object identifier for products, attribute and class records as the master database. As needed, data from the master product administration database must be replicated to other environments, ensuring that the object identifiers are preserved across the databases.
- Never manually create objects with the same name in two databases. They are different objects and do not merge into one record, using the import function or Application Deployment Manager (ADM). If the object identifier is different for the same object in the two databases, the versioned object is treated as a different object during the import even if the names match, hence the two objects are not merged. Instead, an existing object definition can be moved from one database to another using import or ADM, so that the object identifier is preserved.
- The product, attribute, and class master must initially be a copy of the upgraded production database.

How Duplicate Objects Are Created

When you migrate customizable product (CP) data using product export and import or Application Deployment Manager (ADM) and the joint workspace data type, you can create duplicate records in the destination environment database when the CPs in the two environments have the same name but have different object identifiers. The S_PROD_INT.CFG_MODEL_ID column stores the versioned object identifier.

For example, assume that the source environment includes the product records shown in Table 41, and the target environment includes the product records shown in Table 42.

Table 41. Example of Data in Source Environment

Product Name	Row ID	CFG_MODEL_ID
Product A	1-A	1-A
Product B	1-B	1-B

Table 42. Example of Data in Target Environment

Product Name	Row ID	CFG_MODEL_ID
Product A	1-A	1-A
Product B	1-C	1-C

After the data migration, the target application includes the records shown in Table 43.

Table 43. Example of Data in Target Environment After Migration

Product Name	Row ID	CFG_MODEL_ID
Product A	1-A	1-A
Product B	1-C	1-C
Product B (1-B)	1-B	1-B

Product A is imported correctly to the target environment, because it has the same identification number in both environments.

Product B has different identification numbers in the two environments. Therefore, when it is imported, a duplicate product with the name Product B (1-B) is created. Product B (1-B) contains the latest changes of the product that were made in the source environment, while the original Product B in the target environment is unchanged.

NOTE: All run-time data, such as quote line items, order line items, and assets, still refer to the original Product B. If a user customizes the product, Siebel Configurator will use the obsolete product definition based on the original Product B rather than the updated product definition of Product B (1-B).

Possible Symptoms of Duplicate Records

The following symptoms probably mean you have duplicate records:

- The original versioned object definitions are not updated.
- New object definitions with the imported object identifier are created and have a long name made up of the product name and row ID.
- The quote and order transaction data refer to outdated product definition.
- If you customize these products in a quote or order record, the quote uses the outdated definitions instead of the updated product definitions that you imported.
- Duplicate product, attribute and class records are in the target environment.

Name-Based Import

If you have failed to create a master product database and you already have multiple databases with different object identifiers for the same objects, object identifier-based import fails. In this case, you can use name-based import. Name-based import is not recommended. Use it only as a last resort.

Name-based import is done using the ISS Authoring Import Export business service. For more information, see *Siebel Order Management Guide*.

Creating Time Slice Reports for Product Versions

You can produce time slice report for a customizable product versions. This report lists dependent objects that expire prior to the end date of the version.

If dependent objects expire, the product version may not work as intended. After viewing the time slice report, if necessary, you can change the end dates of existing versions of the dependent objects or create new versions of the dependent objects with later end dates.

The time slice report produced in Customizable Products, then Versions view only takes account of objects that have already been released. To produce the report for objects that have not been released, you must use Workspace Projects view and select the Use Project checkbox for the objects.

The time slice application preference determines the output format of time slice reports (XML or CSV). For information about setting application preferences, see *Siebel Applications Administration Guide*.

To create a time slice report for objects that have been released

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the desired customizable product.
3. In the Versions list, select the version you want to run the report on.
4. From the Versions list menu, select Create Time Slice Report.

To create a time slice report for objects that have not been released

1. Navigate to the Administration - Product screen, then the Workspace Projects view.
2. In the Workspace Projects list, add a new record and complete the necessary fields.

   The Use Project checkbox must be selected.
3. In the Contents view, add all the objects that you want to run the report on.
4. From the Workspace Projects list menu, select Create Time Slice Report.

Releasing Products for Use

To make a customizable product available to users, you must release it. Releasing a customizable product creates a new version of it and makes this version available for use, if versioned data was changed.

If no versioned data was changed, a new version of the product is not created when you release the product. However, if the release date falls between the existing versions, a new version record is created even if no versioned data was changed.

You cannot modify or delete a released version of a customizable product. After you release a version, the workspace retains the image of that version, so it can be used as a starting point for the next version.

Before you release a customizable product, whether it is available to users depends on these fields:

- **Start Date.** When you release a product, it becomes available to users on the start date, and it remains available until the next start date of a version of the product. Start date is precise to the minute and second. For example, if you release a product with the start date of January 1, 2006 12:00:00 and another released version of the same product has the start date of July 1, 2006 12:00:00, then the newly released version will be available to customers from January 1 12:00:00 through June 30, 2006 12:00:00. This lets you release several versions of a product and have them become available based on dates.
- **End Date.** The end date for a version is entered automatically, based on the start date for the version that is next chronologically. This local date and time are converted to Greenwich Mean Time, so the version is available at the same times for all servers connected to this database.
- **Active.** When you release a product, it is only available to users if the Active checkbox of the current version is selected.

When remote users synchronize databases, they receive all released versions of a customizable product not already on the local computer.

Observe the following guidelines when using start dates for products with components:

- Carefully coordinate configuration rules that have effective dates with the start date of the customizable product. If you know that you must release a new version of the product because of time-sensitive configuration rules, consider using a version start date rather than effective dates on configuration rules.

- If you want to release two versions such that the second version supersedes the first version on its start date, be sure to fully analyze the possible business impacts of the new version on in-process, and existing quotes.
- The start date cannot be modified after a version is released. Instead, you can release a new version to replace or intercede an existing version.

To release a customizable product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired customizable product.
3. In the Versions list, enter a date in the Start Date field of the Work Space record.
4. Click Release to release a new version of the product.

   A new record appears in the Versions list. Its version number displays in the Version field. The Required State Date field becomes read-only.

Deleting Product Versions

If you no longer want to use a version, you can release a new version to replace the version you do not want to use, as described in "Replacing Earlier Product Versions" on page 348.

If you have a large number of inactive versions for a given product, you can delete the versions using the CleanupSingleObject method of the ISS Authoring Import Export Service. For more information, see *Siebel Order Management Guide*.

Replacing Earlier Product Versions

Product versions are visible to customers on the dates between the values in the start date and the end date. You enter a value in the Start Date field when you create the version. The value in the End Date field is entered automatically, based on the start date of other versions that come after the start date of this version.

If you create a new version with the same start date as an existing version, the existing version will have a end date that is the same as its start date, so it will never be visible to customers. The version is never visible to customers, because there is no time between its start date and its end date.

To replace an earlier version of a customizable product

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select and lock the desired customizable product.
3. Enter a date in the Start Date field of the Work Space record in the Versions list that is identical to the Start Date of the version you want to replace.

   You can copy the start date from the version you want to replace and paste it in to the Work Space record.

4 Click Release to release the Work Space record as a new version of the product.

Displaying Product Versions that Are Available to Customers

When you replace earlier versions of a product, you create version records that are never visible to customers because their start date and end date are the same.

In the Versions list, you can use the Time Filter button to filter out versions that have been replaced and view only versions that will be visible to customers.

This button is a toggle. After you have used it to hide versions that have been replaced, click it again to display all versions.

To display only product versions that are visible to customers

1 Navigate to the Administration - Product screen, then the Product Definitions view.

2 In the Products list, select the desired customizable product.

3 In the Versions list, click Time Filters.

Versions whose Start Date and End Date are identical disappear, so only versions that will be visible to customers appear.

4 In the Versions list, click Time Filters again.

All versions appear.

Making Products Unavailable to Customers

Some products are only available to customers at certain times. For example, a seasonal product used during the summer may be available for purchase only during the months of May through August.

You cannot make these products unavailable by entering a End Date for versions of these products, because values are entered in this field automatically, based on the start date of subsequent versions.

Instead, release a new version of the product with the Active checkbox deselected, so this product is not visible to customers for the duration of this version. The start date for this version will become the end date for the previous version.

For example, for a product that is only available for purchase from May through August:

- Create a version with the start date of May 1, 2006 with the Active checkbox selected.
- Create a version with the start date of September 1, 2006 with the Active checkbox deselected.

This product is only visible between May 1, 2006 and August 31, 2006.

Reverting to Earlier Versions of Products

If you release a customizable product and then make changes to its current work space, you can discard all the changes and revert to a version of the product that you released earlier.

When you revert, the entire contents of the current work space is discarded. This includes the product's structure, attributes user interface, rules, links, resources, and scripts. You choose an earlier version, and an instance of it is then loaded into the current work space.

To revert to an earlier version

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. In the Products list, select the desired customizable product.
3. In the Versions list, select the version you want to revert to.
4. In the Versions list, click Revert To Selected Version.
5. Save the current work space.

Releasing Multiple Products Using Workspace Projects

The instructions in earlier topics of this chapter describe how to release versioned items one at a time. You can also release a number of versioned items at the same time by using the Workspace Projects view.

If you are modifying and releasing products one at a time, use the methods described in the earlier topics of this chapter.

If you are modifying many products and other versioned items as part of one project and you want to release them all at one time, use the method described in this topic.

To release multiple products simultaneously

1. Navigate to the Administration - Product screen, Workspace Projects, and then the Contents view.
2. In the Contents list, select all the items you want to release.
3. Click Release New Version.

To release all locked products

1. Navigate to the Administration - Product screen, Workspace Projects, and then the Contents view.
2. Make sure all objects in the Contents view are locked.
3. In the Contents view, click Release All.

Managing Products Using Workspace Projects

You can also use the Workspace Projects view to manage objects and create relations among them, without having to release them.

To manage objects using Workspace Projects view, perform the tasks in the following high-level process:

- Create any new objects you need in the appropriate views for each object.

  For example, create products in the Administration - Products, then the Products view, product classes in the Administration - Products, then the Product Classes view, and so on.

- Navigate to the Administration - Products, then the Workspace Projects view and add a new record to the Workspace Projects list.
- In the Administration - Products, Workspace Projects, and then the Contents view, create new records and add all these objects to this Workspace Project.
- In the Workspace Project record, select the Use Project check box.

  Selecting this check box activates the Workspace Project and makes the workspaces of these objects visible to each other for the duration of the session, even though the objects have not been released.

- Drill down on the item in the Contents list, and associate the objects to each other, as needed.

  After drilling down on them, the objects appear in the usual views you use to work with them, and you can associate them in the usual ways.

- Test the objects using Scenario Tester, as described in "Process of Testing Products with Scenario Tester" on page 335.

  NOTE: For objects explicitly listed in the active Workspace Project, workspace are used. For all other objects, the released versions active on the test date are used as needed.

- Release the objects together using Workspace Projects, or release the objects individually, as needed.

Migrating Products Among Environments

Some businesses develop new versions of products in the same environment that they use for product that are already in production. Other businesses use separate environments for development and for production.

If you use separate environments for development and production, you can migrate product data from the development to the production environment in either one of two ways:

- "Migrating Products Using Import and Export in Workspace Projects View." This method allows you to move versioned data and related objects.

- "Migrating Products Using Application Deployment Manager (ADM)." This method allows you to move both versioned and unversioned data.

NOTE: You cannot use EAI to import versioned data.

Migrating Products Using Import and Export in Workspace Projects View

Using import and export in Workspace Projects view, you can migrate the versioned data that you can add to the Workspace Projects list, and you have the option of migrating related objects. When you export objects, a dialog box appears with these options:

- **Object(s) Only.** Exports only the objects in the Workspace Projects list.
- **Full Structure.** Exports the objects in the Workspace Projects list and related objects, such as attribute definitions and product classes. If you use this option, the related objects are imported automatically when you import the objects in the Workspace Projects list.

In either case, all the objects are exported to a single XML file. When you import this XML file, they are imported as separate objects.

The data that is exported depends on the value in the Testing Date field and Use Project flag of the Workspace Project header. If no base date is entered, it depends on the value in the Effective Start Date field.

NOTE: To extend the import/export XML schema, extend integration objects under the project ISS VOD Import Export.

Each object is imported in a single transaction. Extend text fields only and avoid extending FK columns or adding child entities. An inconsistent product definition might result in the case of one object failing to import due to an invalid FK or child entity, and other objects import correctly.

It is recommended not to extend integration objects because the preconfigured import is staged in the Workspace version and does not take effect until you release new product versions. However, an extension which is non-versioned takes immediate effect after the import.

The recommended approach is to use preconfigured IO to migrate product headers and versioned objects, and extend the production ADM object to migrate the customer's extension.

To migrate product data using export and import in Workspace Projects view

1. In the development environment, navigate to the Administration - Product screen, then the Workspace Projects view.

2 In the Workspace Projects list, in the record for workspace containing the objects you want to migrate, enter the necessary information in the fields shown in the following table.

Field	Comments
Effective Start Date	This date is used for publishing. If no Testing Date is entered, this value is the date that the application uses to determine what versions must be exported for dependent items when the user chooses to export the full structure of the items in the Workspace Project
Base Date	This date is used for temporarily setting arbitrary time mainly for testing and development when Workspace Project is activated. Enter the date used as the date for dependent items for full structure export. If both Effective Starting Date and Base Date are filled, Base Date is used. If you are exporting from an active Workspace Project (Use Project is checked): ■ All root objects, listed in the content, use Workspace Project versions ■ All referenced objects (resolved through dependencies) use a published Workspace Project version effective for Base Date If you are exporting from a non-active Workspace Project (Use Project flag is not checked): ■ All root objects, listed in the content, use a published Workspace Project version effective for Base Date ■ All referenced objects (resolved through dependencies) use a published Workspace Project version effective for Base Date If Base Date is not specified, Effective Start Date is used. If both Effective Start Date and Base Date are not specified, then current time is used.
Use Project	If Use Project is checked, the workspaces of the objects in the Contents list are exported. If Use Project is not checked, the relevant versions of those objects are exported.

3 In the Workspace Projects menu, select Export Contents.

4 In the Export Versioned Object dialog box, click Object(s) Only or Full Structure.

5 Save the files that are exported.

6 In the target environment, navigate to the Administration - Product screen, then the Workspace Projects view.

7 In the Workspace Projects list, add a new record and complete the necessary fields.

The contents will be imported into this new Workspace Project.

8 In the Workspace Projects menu, select Import Contents.

9 In the VOD Import dialog box, click Browse and select the file to import, and then click Import.

The imported file appears in the Workspace Projects list.

Migrating Products Using Application Deployment Manager (ADM)

You can also migrate products using Application Deployment Manager. You use the Joint WorkSpace content object to migrate key fields and versioned content for product itself, and you use the Product (ADM) content object to migrate the non-versioned content of product.

If you are migrating non-standard fields, you must customize the Product (ADM) content object.

For information about how to use ADM, see *Siebel Application Deployment Manager Guide*.

Importing and Exporting Product Promotions

Product promotions can be imported and exported as follows:

- Navigate to the Administration - Product screen, then the Workspace Projects view.
  - In the Workspace Projects menu, select Export Contents.

  The Export Full Structure option invokes method ExportFullVod of the ISS Authoring Import Export Service.
- Navigate to the Administration - Product screen, then the Product Promotions view.
  - In the Product Promotions menu, select XML Export option.

  In this case, method ExportVOD of ISS Authoring Import Export Service is invoked.

  NOTE: The XML file generated contains the definition of the promotion. Only one promotion can be exported at a time.

For more information about methods of the ISS Import Export Authoring Service, see *Siebel Order Management Guide*.

22 Product and Promotion Eligibility and Compatibility

This chapter describes eligibility and compatibility rules for products and product promotions. It includes the following subjects:

About Product and Promotion Eligibility

Product and promotion eligibility allows you to create rules specifying which customers are eligible to buy a product and which customers are eligible for product promotions.

NOTE: Eligibility supports product promotions, which are created in the Promotions view of the Administration - Product screen, but not other types of promotions. For more information, see the topic about product promotions in *Siebel Pricing Administration Guide*.

Some examples of product eligibility are:

- A telecommunications company may have different wireless plans available for different geographical areas.

- An internet service provider may have certain DSL services only to customers who live within a specified distance of certain cities.

Some examples of promotion eligibility are:

- An airline may offer a free upgrades only to customers who live in certain cities.
- A cable television company may offer extra channels at no extra cost only to new customers who live in certain states.

Eligibility is based on the PSP engine, so it is fully configurable. For more information about PSPs, see *Siebel Order Management Infrastructure Guide*. For more information about the PSP used for eligibility, see "Eligibility and Compatibility Workflow Reference" on page 378.

About Eligibility Rules and Configuration Constraints for Siebel CRM Version 7.7 and Earlier

In versions Siebel CRM 7.5 and 7.7, you may have used configuration constraints, linked items, and scripting to create rules controlling which customers could buy a product. In Siebel CRM version 7.8 and later, you can do this much more efficiently using eligibility rules.

For example, you want to limit the availability of a product named 104 USB Keyboard to customers who have a type of Commercial:

- In Siebel CRM version 7.5, you may have done this by creating a linked item to bring the customer type into the configuration session. Then you write Siebel Configurator exclude constraints to exclude this product if the type is not Commercial. You must write these rules many times, once for every product that can use this keyboard as a component.
- In Siebel CRM version 7.5, you may also have done this by assigning all component products are assigned a Siebel Configurator UI property named Commercial Only with values of Yes or No. Then, you modify the JavaScript code that displays the available selections to evaluate the condition: Is customer type NOT "Commercial" and does this "Commercial Only" UI property for this product equal "YES." If this condition is true, the JavaScript does not display that product or displays it differently.
- In Siebel CRM version 7.8 and later, you can do this by creating an eligibility rule. You write this rule once, and it applies to all configuration models. By setting the Eligibility Display Mode, you can control whether ineligible products are displayed in red, as a warning, or not displayed at all. You do not have to use any scripting.

Without configuration, compatibility rules do not necessarily substitute for Siebel Configurator requires and excludes rules. The difference is that the compatibility rules would work against the Projected Asset Cache, which includes all assets for the customer, all open orders for the customers and the current quote for the customer. This broad scope may be unacceptable if you want the scope of the rule to just be the components within one customizable product instance, as it is for configuration constraints. For more information about the Projected Asset Cache, see *Siebel Order Management Infrastructure Guide*.

By default, the Projected Asset Cache, with its extended scope, is used in workflows, whereas Siebel Configurator rules apply only to the loaded product. If you want workflows to have a more limited view, identical to Siebel Configurator rules, you must change the default Eligibility and Compatibility Procedure by modifying the Projected Asset Cache query. As an example, see the step Initialize PAC and the following steps in "Eligibility and Compatibility Workflow Reference" on page 378.

However, if you are using linked items and attributes and configuration constraints to control product eligibility, then eligibility rules are a preferred replacement for these configuration constraints.

By using eligibility rules instead of the modeling approaches used in earlier versions, you achieve:

- Greater scalability
- Faster performance
- Simplified administration
- Less customization, easier upgrades
- More consistent behavior across the application

Defining How Eligibility Output Displays

Eligibility is checked before the product or promotion is displayed in a catalog, Product picklist, or elsewhere in the interface.

Eligibility information is displayed when you select products to add to quotes, orders, or agreements. You can define how this information is displayed by setting the Eligibility Display Mode parameter value. Setting this value controls the Eligibility and Compatibility procedure which dictates the behavior of eligibility and compatibility as follows:

- If set to 0, the procedure is not run. This stops price list visibility, contract visibility, and product effectivity filters from running also.
- If set to 1, the procedure is run, and ineligible products are displayed with messages. In the Siebel Configurator, ineligible products are displayed in red text.
- If set to 2, the procedure is run, and ineligible products are not included in selections.

You can set the Eligibility Display Mode parameter value at various levels in your application. For example, you can choose to set the parameter value from the Enterprise, Server or Object Manager level. You can also set the value at the applet, workflow call, or at the data service API level.

The Eligibility and Compatibility procedure looks for the display mode value in the following locations and in the following order:

1. Display Mode workflow process property
2. The EligibilityDisplayMode property on the applet
3. The Eligibility Display Mode server parameter
4. The Eligibility Display Mode default value (1 – show all products with messages)

Eligibility Display Options for Catalogs

In catalogs, there are three options for how eligibility information is displayed:

- **Show Products Only.** All products are displayed, whether or not the user is eligible to purchase them. Eligibility information is not displayed to the user. This option also turns off product effectivity, price list and contract visibility.
- **Hide Products.** Products are not displayed if the user is not eligible to purchase them.
- **Show Products with Comments.** All products are displayed. A comments displayed with each product says whether the user is eligible to purchase it.

If customers are eligible for only a small percentage of the total products, then choosing the Hide Products option gives slower performance than the Show Products with Comments option. This is because a large number of records must be read before enough records are found to fill a page. This performance trade-off must be considered when selecting the Display Mode during implementation design.

Eligibility Display in Siebel Configurator

In Siebel Configurator, if the customer is ineligible to buy the product, then the application displays an error message if the user selects the product. The user can click Undo to undo the selection that caused the error, or select Proceed to go ahead with the configuration using the ineligible product. If the user selects Proceed, Siebel Configurator adds this error message to its message list.

NOTE: In addition to being set in the applet, the eligibility setting can be set by a workflow or by a server parameter.

Siebel Configurator also enforces the value set for the Display Mode parameter. If this parameter is set not to show ineligible products, then the ineligible products are not in the configuration session, and the user cannot choose them.

To define how eligibility output displays on the mobile or dedicated client

1. Use any text editor to edit the CFG file for the application.
2. In the [PSPEngine] section, set the value of the parameter EligibilityDisplayMode to one of the following values:
   - **EligibilityDisplayMode=0.** No eligibility comments are displayed, and all products (eligible or not) appear in all product selection applets
   - **EligibilityDisplayMode=1.** Default value. Ineligible products appear in the product selection applets with the eligibility status of N and with comments explaining why they are ineligible.
   - **EligibilityDisplayMode=2.** Only eligible products appear in the product selection applets, so the user cannot see and pick ineligible products.

To define how eligibility output displays on a server-based application

- Modify the value of the server or component parameter named EligibilityDisplayMode, using the same values as in the previous procedure.

Defining Eligibility Groups

Eligibility groups are used to group eligibility rules that you create, for your own purposes.

For example, you can create one group for product eligibility rules and one group for promotion eligibility rules. Alternatively, you may decide to create one group for each product line. You can use any logical group that makes sense to your business.

NOTE: Whatever the grouping, when eligibility is evaluated, all rules of type *eligibility* will be evaluated against all products, regardless of how the products are logically grouped into Eligibility Groups. The same is true of rules of type compatibility.

Choose how to group eligibility rules based on your business needs.

To define an eligibility group

1. Navigate to the Administration - Product screen, then the Eligibility and Compatibility Matrices view.
2. In the Eligibility and Compatibility Matrices list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for a group.
Matrix Type	To make this an eligibility rule group, select Eligibility Rules.

Defining Product and Promotion Eligibility Rules

Before you can define product eligibility, you must define the product. Then add eligibility rules to it.

Before you can define promotion eligibility, you must define the product promotion, as described in the topic on product promotions in *Siebel Pricing Administration Guide*. Then add eligibility rules to it.

You can create two types of rules:

- **Available Rules.** specify the customers who are eligible to buy the product.
- **Not Available Rules.** specify the customers who are not eligible to buy the product.

Defining Inclusive or Exclusive Eligibility for Products

A given product can use only inclusive or exclusive eligibility rules. Before you define the rule, you must define which type of eligibility the product uses.

To define inclusive or exclusive eligibility for a product

1. Navigate to the Administration - Product screen, then the Products view.

2 In the Products list, in the record for the product, select or deselect the Inclusive Eligibility checkbox, if necessary.

Defining Product Eligibility Rules

You can define eligibility rules for products or product lines.

This procedure describes how to define rules for products. To define rules for product lines, use a similar procedure, but instead of adding a record to the Eligibility Rules list, add a record to the Product Line Eligibility list, which is further down on the screen.

To define product eligibility rules

1 Navigate to the Administration - Product screen, then the Products view.

2 In the Products list, select the product you are defining rules for.

3 Click the Eligibility and Compatibility Rules view tab.

4 In the link bar of the Eligibility and Compatibility Rules view, click Eligibility.

5 In the Eligibility Rules list, add a new record and complete the necessary fields, described in the following table.

Field	Comments
Matrix Name	Select the eligibility group used for grouping this rule. For more information about eligibility groups, see "Defining Eligibility Groups" on page 359.
Rule Type	Select the type of the rule. Options are: ■ **Available.** The rule specifies customers who are eligible to buy. For example, if the criteria depends on State, the rule specifies a state where customers are allowed to buy the product. ■ **Unavailable.** The rule specifies the customers who are not eligible to buy. For example, if the criteria depends on State, the rule specifies a state where customers are not allowed to buy the product.
Account	If the rule depends on account, select the name of an account that is or is not eligible to buy the product.
Account Type	If the rule depends on account type, select the name of an account that is or is not eligible to buy the product.
City	If the rule depends on city, enter the name of a city where customers are or are not eligible to buy the product.
State	If the rule depends on state, select the name of a state where customers are or are not eligible to buy the product.

Field	Comments
Country	If the rule depends on country, select the name of a country where customers are or are not eligible to buy the product.
Postal Code	If the rule depends on postal code, enter the name of a postal code where customers are or are not eligible to buy the product.
Effective Start	Enter the date when this rule goes into effect.
Effective End	Enter the date when this rule is no longer in effect.

6 Continue to add records until you have specified all eligibility rules that determine whether customers are eligible to buy this product.

Defining Promotion Eligibility Rules

This procedure describes how to define eligibility rules for product promotions.

To define promotion eligibility rules

1 Navigate to the Administration - Product screen.

2 In the link bar, select Product Promotions.

3 In the Product Promotions list, select the promotion you are defining rules for.

4 Click the Eligibility and Compatibility Rules view tab.

5 In the link bar of the Eligibility and Compatibility Rules view, click Eligibility.

6 Add records to the eligibility rules list and enter criteria in the same way that you do when you are defining product eligibility rules.

Defining Eligibility for Products with Components and for Component Products

Eligibility for products with components is checked at the following times:

- **Product Selection.** When a new product with components is selected, the application checks the customer's eligibility to buy the product with default options. For example, eligibility is checked when an eSales user clicks Add to Cart.
- **Siebel Configurator Launch.** When the customer launches a new configuration session, the application checks the customer's eligibility to put the product in the saved quote, since eligibility may have changed since the quote was saved.
- **Component Products.** There is an eligibility check on component products that the customer can add to the configurable product.

Eligibility for products with components is defined in the same way as eligibility for other products:

- You can define eligibility for individual component products of the product with components, as described in "Defining Compatibility Rules for Products and Promotions" on page 373.
- You can define eligibility for the entire product with components, as described in "Defining Compatibility Rules for Products and Promotions" on page 373.

If you define eligibility for component products, the customer may not be eligible to buy all the default components of a product with components but may be eligible to buy the product with components with substitute components. In this case, the product with components can be flagged as ineligible, so customers can select substitutes for the default components that they are not eligible to buy.

Defining Eligibility for Product Attributes

You can define the eligibility for individual attributes of products. For example, only customers living in certain countries are eligible to buy a shirt in a specific color. All customers are able to buy the same shirt in any other color.

When a configuration session is started, all the possible attribute values belonging to the attributes marked as requiring an eligibility check are evaluated:

- If the eligibility display is Mode 1, the invalid attribute values are displayed in red within the configuration session.
- If the eligibility display is Mode 2, the invalid attribute values are not displayed in the configuration session UI (unless they are already selected).

When an attribute value is changed, only the new selection is evaluated for eligibility (post-pick). The ineligibility reason is used by the Siebel Configurator UI in the messaging.

You must create custom PSP procedures to evaluate the attributes. The framework that calls these custom procedures and communicates with the user is delivered as part of the application, including:

- Ability to assign an evaluation procedure to an attribute.
- Evaluation of attribute eligibility within a configuration session.
- Creation of context and row set data used by the custom PSP procedure.
- Attribute eligibility related messaging within a configuration session. If the user selects an invalid attribute, Siebel Configurator displays a message telling the user to click the Undo button.

Saving the Result of Attribute Eligibility

Attribute eligibility status and reason are not saved to the quote or order objects when a configuration session is completed. It is not possible to save these with the configuration changes alone, because the values are not saved to the CxObject in memory.

However, it is possible to save these results using the BatchValidate API for Siebel Configurator, described in "BatchValidate Method" on page 463. This API can run attribute eligibility. The ineligible attributes are sent to a log file. The output property that is set from this API contains information on the ineligible attributes.

It is possible to add a custom step to the Sales Order verify process that calls the batch validate API. This step can update the custom fields on the Order XA business component indicating an attribute value as being ineligible. The Order Validation engine then checks that field to confirm that no ineligible values are selected.

Attribute Eligibility Architecture

Attribute eligibility checking uses the following architecture:

- For attributes requiring an eligibility check, the user creates a custom PSP procedure and attaches it to this attribute when it is defined on the class.
- When the configuration session is started, for each attribute that will be evaluated for attribute eligibility, a context row set, row set, and additional parameters are generated, and then the custom PSP procedure is called.
  - The context row set is based on the variable map Cfg Eligibility Variable Map - Context.
  - The row set is a list of all possible values for the attribute.

  NOTE: When an attribute value is changed, the same events occur except that the only value evaluated is the new value selected.
- The custom procedure evaluates each value in the row set to find values that are ineligible, and the custom procedure creates the following variables with the following values:
  - Eligibility Status = N
  - Eligibility Reason = user defined message
- With the default Eligibility Display Mode of 1, the Siebel Configurator UI uses this status to determine which selections are displayed in red.
- If Eligibility Display Mode was set to 2, then the ineligible values are not displayed.
- The Siebel Configurator UI uses the Eligibility Reason in the message displayed when the end user selects an invalid attribute value.

Setting Up Attribute Eligibility Checking

This topic describes the high-level steps for setting up attribute eligibility checking.

Writing a Custom PSP Procedure to Check Eligibility

You can use a given procedure to evaluate one attribute or many attributes:

- Use conditional statements to use one procedure to evaluate multiple attributes that have different evaluation criteria.
- Or create one procedure for each attribute.

Typically, the custom procedure includes the following steps:

- In a step of the PSP procedure, log the Context property set.
- In a step of the PSP procedure, log the RowSet property set.

- In a step of the PSP procedure, set all rows to Eligible by default before evaluating the row set:
  - This step uses the Row Set Transformation Toolkit business service, Conditional Action Transform Method.
  - Input arguments are shown in the following table.

Input Argument	Type	Value	Property Name
On Default 1	Literal	{Row.Eligibility Status} = LookupValue('PEC_ELIG_STATUS_CD','Y')	
Row Set	Process Property		Row Set

  - Output arguments are shown in the following table.

Property Name	Type	Value	Output Argument
Row Set	Output Argument		Row Set

- In one or more steps of the PSP procedure, check attributes for eligibility:
  - This step uses the Row Set Transformation Toolkit business service, Conditional Action Transform method.
  - Input arguments include conditional statements that check whether the value entered is eligible.
  - Output arguments set rows to ineligible if these conditions are true.
- In Administration – Order Management, then the Message Types view, define the error message.
- In Administration – Order Management, then the Payload view, create payload variable records.
- In a step of the PSP procedure step, assign context and rowset values to the message payload variables.

Creating the Attribute and Assigning It to a Product Class

Create the attribute and assign it to a product class in the usual way, described in "Process of Creating Products with Attributes" on page 65.

In the Administration - Product, Product Classes, Version, and then the Attributes view:

- In the Check Eligibility field, select the checkbox for any attribute whose eligibility you want to check.
- In the Eligibility Procedure field, select the custom PSP procedure that you created to check the eligibility of this attribute.

Changing the Context and Rowset Variable Maps

You can change the context and rowset variable maps used by the attribute eligibility procedure. The variable map is set as a parameter in the LoadInstance signal and the SetInstance signal. The name of the parameter is Context Variable Map. By default, it has a value of Cfg Eligibility Variable Map – Context.

The rowset for attribute eligibility is generated without a variable map. It contains only one property: value.

Using Different Display Modes for Attribute and Product Eligibility

You can have different display modes for Attribute eligibility and for Product eligibility. Change the display mode for attribute eligibility by adding the EligiblityDisplayMode parameter to the SetInstance and LoadInstance signals and giving it the value of 0, 1, or 2. This overrides the application's EligibilityDisplayMode setting.

Creating Eligibility Matrices

When you create an eligibility matrix, you enter data similar to the data entered in an eligibility rule, but you enter the data under the matrix.

You must use eligibility to create eligibility rules for product classes.

Eligibility and compatibility rules are shown in the Eligibility and Compatibility Matrices view, regardless of how you created them. If you create them in Eligibility and Compatibility Rules view and grouped them under a matrix, the rules are displayed under this matrix. This procedure describes how to create them directly under the matrix.

To create an eligibility matrix

1. Navigate to the Administration - Product screen, then the Eligibility and Compatibility Matrices view.
2. In the Eligibility and Compatibility Matrices list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for a group.
Matrix Type	Select Eligibility Rules.

3. Click the name of the new record.

   The Eligibility Matrix form and Eligibility Rules list appear.

4 Add a new record to the Eligibility Rules list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Product	Select the subject product. **NOTE:** You may use only one field of Product, Product Line, and Product Class.
Product Line	Select the subject product line.
Product Class	Select the subject product class.
Type	Select the type of the rule. Options are: ■ **Compatibility - Requires.** If the subject product is purchased, the object product must be purchased. ■ **Compatibility - Excludes.** If the subject product is purchased, the object product must not be purchased.
Account, Account Type, City, State, Postal Code, Country	In one of these fields, enter the account, account type, city, state, postal code, or country that the rule applies to. Depending on the Rule Type that you chose, this is required to buy the product or excludes you from buying the product.
Effective Start	Enter the date when the rule goes into effect.
Effective End	Enter the date when the rule is no longer in effect.

About Product and Promotion Compatibility

Compatibility allows you to define rules specifying which combinations of products, product attributes, or product promotions are required and which combinations are not allowed. These rules are simpler than configuration constraints and can be defined by the marketing administrator.

NOTE: Compatibility supports product promotions, which are created in the Promotions view of the Administration - Product screen, but not other types of promotions. For more information, see the topic about product promotions in *Siebel Pricing Administration Guide*.

Compatibility rules are global. Once they are defined for a product or promotion, they apply whenever it is added to quotes, orders, or agreements.

If the end user selects a product that has a compatibility rule associated with it, the application displays a message describing the rule. The message may say that the selected product requires purchase of another product or that the selected product is not compatible with a previously selected product.

The application does not enforce compatibility rules, as it enforces configuration constraints. It just displays these messages.

Eligibility and compatibility checks are performed by the same workflow and the same engine at the same time. Compatibility is one step in the overall determination of eligibility. Compatibility rules are just one of a number of different criteria that the eligibility procedure checks to determine whether a product can be sold to a customer given the customer's profile.

Compatibility is based on the PSP engine, so it is fully configurable. For more information about PSPs, see *Siebel Order Management Infrastructure Guide*. For more information about the PSP used for compatibility, see "Eligibility and Compatibility Workflow Reference" on page 378.

About Compatibility Rules

Compatibility rules state that products require or exclude other products. You can create rules for products, product lines, and product classes.

As a simple example, assume that customers who buy a computer must buy a monitor, and there are only two monitors available. The Turbo computer supports the Standard monitor but not the Flat-screen monitor. Then you would create two different rules:

- Turbo computer requires Standard monitor.
- Turbo computer excludes Flat-screen monitor.

The Subject and Object of Compatibility Rules

As you can see in this example, compatibility rules consist of:

- **Subject.** The product that requires or excludes another product. In the examples, the computer is the subject.
- **Type of Rule.** The type is either *requires* or *excludes*.
- **Object.** The product that is required or excluded. In the examples, the monitor is the object.

Requires rules apply in only one direction. If the subject requires the object, the object does not necessarily require the subject. For example, the Turbo computer requires a Standard monitor. However, the Standard monitor does not necessarily require the Turbo computer; it can also be used with other computers.

Excludes rules apply in both directions. If the subject excludes the object, the object necessarily excludes the subject. For example, the Turbo computer does not work with the Flat-screen monitor, and that necessarily means that the Flat-screen monitor does not work with the Turbo computer.

Subject Compatibility Rules and Object Compatibility Rules

For product compatibility:

- When you create a subject compatibility rule, you choose a product as the subject of the rule, and you can choose either a product, product line, or product class as the object of the rule.
- When you create an object compatibility rule, you choose a product as the object of the rule, and you can choose either a product, product line, or product class as the subject of the rule.

For product line compatibility:

- When you create a subject compatibility rule, you choose a product line as the subject of the rule, and you can choose either a product, product line, or product class as the object of the rule.
- When you create an object compatibility rule, you choose a product line as the object of the rule, and you can choose either a product, product line, or product class as the subject of the rule.

About Pre-Pick Compatibility

By default, compatibility rules are evaluated after the sales agent selects a product. This is called post-pick compatibility.

It is also possible to enable pre-pick compatibility, so the sales agent can see whether products are compatible before selecting them.

If pre-pick compatibility is enabled, products are displayed as follows:

- **Catalog.** If a product displayed in the catalog is incompatible with a product already added to the order, a compatibility message is displayed under the product name.

  For example, a customer orders satellite dish programming service. She tells the sales agent that she wants the high-resolution satellite dish. Then, in the same order, she wants to purchase a digital video recorder (DVR). Only three of the five DVRs that the company sells are compatible with the high-resolution satellite dish. When the sales agent displays the catalog page listing DVRs, the two DVRs that are not compatible have messages under them that say their compatibility status is N and that also display the compatibility reason explaining why they are incompatible. The sales agent is able to tell the customer about the DVRs that are compatible with the satellite dish, without wasting the customer's time by talking about DVRs that are incompatible.

- **Siebel Configurator.** If a component displayed in Siebel Configurator is incompatible with a component already selected, then it is displayed in red to indicate it is excluded. The sales agent can select the excluded components to display the compatibility message explaining why it is excluded.

  For example, a cable television customer orders a cable television subscription and selects the standard package of channels. The sales agent who is configuring the service that the customer is purchasing selects these two products in Siebel Configurator. Then the customer says that he also wants to subscribe to a special programming service that provides additional channels, which he saw in an advertisement. In Siebel Configurator, this special package is displayed in red, indicating that it is excluded. The sales agent selects the excluded product, and Siebel Configurator displays an error message. The sales agent clicks the explanation icon, and the explanation says that this special programming service is not available with the standard package of channels, only with the premium package.

When pre-pick compatibility is enabled, the application still does the post-pick compatibility check. For example, pre-pick compatibility is enabled and a product in Configurator is marked in red to show that it is incompatible with the current selections. The user adds this incompatible product. The application displays a post-pick compatibility error message saying that this product is incompatible.

Enabling Pre-Pick Compatibility

By default, post-pick compatibility is used. To enable pre-pick compatibility, perform the following tasks:

- "Modifying the Web Template to Enable Pre-Pick Compatibility" on page 369
- "Setting the Server Parameter to Enable Pre-Pick Compatibility" on page 370
- "Setting the Server Parameter to Enable Pre-Pick Compatibility on the Siebel Developer Web Client" on page 371
- "Verifying the Variable Map for Pre-Pick Compatibility" on page 371

For more information about pre-pick and post-pick compatibility, see "About Pre-Pick Compatibility" on page 368.

Modifying the Web Template to Enable Pre-Pick Compatibility

To enable pre-pick compatibility, you must modify the Web template to include the parameter ForceRefresh=Y.

To do this:

- Manually set the Web template for the UI control.
- Edit the Web template to set the ForceRefresh flag to Y.

To manually set the Web template for the UI control

1. Navigate to the Administration - Product screen, then the Product Definitions view.
2. Modify the product definition:
   a. Select the Locked Flag check box for the product.
   b. Drill down on the Workspace version.
3. Click the User Interface Tab.
4. In the User Interface list, add a new record and complete the necessary fields.
5. Drill down on the Name field of the User Interface record you just created.
6. In the Group Item List, add a new record and complete the necessary fields.
7. In the tree structure, right click any item and add select Add Item to add it to the group.
8. For the item you added, select the desired UI Control Web template.

   For example, you might select eCfgControlCheckPriceJS.swt for a check box control.

To edit the Web template to set ForceRefresh to Y

1. Navigate to the directory *siebel install directory*\webtempl/

2 In that directory, find the Web template file for the UI control and open it using a text editor such as Notepad.

For example, you might use Notepad to edit the file eCfgControlCheckPriceJS.swt.

NOTE: The Web templates for controls all begin with eCfgControl.

3 Edit the file:

a Search for the tag <swe: control>.

b Add a line that says: ForceRefresh=“Y”.

c Save the changes.

The edited file has the following code:

```
<!-- Template Start: eCfgControlCheckJS.swt -->
<swe:include file="eCfgPort_HeaderJS.swt"/>


<tr>
   <td colspan=3>
      <swe:for-each id="500" CfgLoopType="DomainAndChildren" startValue="1500" count="Dynamic"
iteratorName="101Id"
       Usage="CheckBox"
       CfgFieldList="CfgFieldName:Name, CfgUIControl:lblName, HtmlAttrib_width:310, Default:Y*
                     CfgFieldName:RequireMoreChild, Default:Y*
                     CfgFieldName:Explanation, CfgUIControl:lblExplanation*
                     CfgFieldName:Customize, CfgUIControl:lblCustomize"
      >
      <swe:control id="swe:111Id + 4000" CfgHtmlType="CfgCheckBox"  ForceRefresh= "Y"
           CfgJSShow="showCheckBox" CfgJSUpdateExclusion="updateExcludedItemForPort"
CfgJSUpdateSelection="updatePortItemsForCheckBox"/>
      </swe:for-each>
   </td>
</tr>
<swe:include file="eCfgPort_FooterJS.swt"/>
<!-- Template End: eCfgControlCheckJS.swt -->
```

Setting the Server Parameter to Enable Pre-Pick Compatibility

Use the following procedure to set the server parameter to enable pre-pick compatibility.

To enable pre-pick compatibility

1 Navigate to the Administration - Server Configuration screen, then the Servers and Parameters view.

2 In the Server Parameters list, query to find the parameter Order Management - PrePick Compatibility.

3 In the Value field for this parameter, enter *True*.

The Value on Restart field also changes to *True*.

Setting the Server Parameter to Enable Pre-Pick Compatibility on the Siebel Developer Web Client

Use the following procedure to set the server parameter to enable pre-pick compatibility on the Siebel Developer Web Client.

To enable pre-pick compatibility

1 Use any text editor to open the .cfg file of the Siebel Developer Web Client.

2 If the InfraObjMgr section already exists, add the following line to that section:

```
PrePickCompatibilityEnabled = TRUE
```

3 If the InfraObjMgr section does not already exist, add both the section header and the line to the .cfg file, as follows:

```
[InfraObjMgr]

PrePickCompatibilityEnabled = TRUE
```

Verifying the Variable Map for Pre-Pick Compatibility

Verify that the needed variable map definition is present for the following two variable maps:

- Product Eligibility Variable Map - Context
- Cfg Eligibility Variable Map - Context

To verify the variable map

1 Navigate to the Administration - Order Management screen, then the Variable Maps view.

2 Query to find the variable map named Product Eligibility Variable Map - Context.

3 Drill-down on this variable map.

4 In the Variable Map Definitions list, query to find Pre Pick Compatibility.

5 If this definition is not found:

a Lock the variable map.

b Add a new record to the Variable Map Definitions list and complete the necessary fields, described in the following table.

Field	Value
Variable Name	Pre Pick Compatibility
In/Out	In

Field	Value
Type	Boolean
On Null	Ignore

c Add a new record to the Variable Source list and complete the necessary fields, described in the following table.

Field	Value
Mode	Any
Path	PARAM_OM_PREPICK_COMPATIBILITY_FLAG
Source Type	Server Parameter

d Add another new record to the Variable Map Definitions list and complete the necessary fields, described in the following table.

Field	Value
Variable Name	Pre Pick Mode
In/Out	In
Type	Text
On Null	Ignore

e Add a new record to the Variable Source list and complete the necessary fields, described in the following table.

Field	Value
Mode	Any
Path	$Current/Header/Pre Pick Mode
Source Type	Instance

f Release the variable map.

6 Repeat Step 1 to Step 5 for the variable map Cfg Eligibility Variable Map - Context.

7 Clear the Cache.

Defining Compatibility Groups

Compatibility groups are used to group compatibility rules that you create, for your own purposes.

For example, you can create one group for product compatibility rules and one group for promotion compatibility rules. Alternatively, you can create one group for each product line. You can use any logical group that makes sense to your business.

NOTE: Whatever the grouping, when compatibility is evaluated, all rules of type *compatibility* will be evaluated against all products, regardless of how they are logically grouped into Compatibility Groups. The same is true of rules of type *eligibility*.

To define an compatibility group

1. Navigate to the Administration - Product screen, then the Eligibility and Compatibility Matrices view.
2. In the Eligibility and Compatibility Matrices list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for a group.
Matrix Type	To make this an compatibility rule group, select Compatibility Rules.

Defining Compatibility Rules for Products and Promotions

You can define compatibility rules for products, product lines, product classes, or promotions.

NOTE: If you create compatibility rules for product lines, they only apply to the primary product line of each product.

All of these types of compatibility rules allow subject compatibility rules and object compatibility rules. For more information about subject and object compatibility rules, see "About Compatibility Rules" on page 367.

Defining Compatibility Rules for Products, Product Lines, or Product Classes

Before you can define a product compatibility rule, you must define the products, and if necessary, the product lines and product classes that it applies to, as described in earlier chapters of this book.

To define compatibility rules for a product

1. Navigate to the Administration - Product screen.
2. Click the Eligibility and Compatibility Rules view tab.
3. In the Eligibility and Compatibility Rules link bar, click Product Compatibility.

4 If you are creating a rule that has the current product record as the subject, then add a new, add a new record to the Subject Compatibility Rules list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Subject Product	Select the subject product. The default is the product that is selected in the Products list, but you can select any product to replace it.
Type	Select the type of the rule. Options are: ■ **Compatibility - Requires.** If the subject product is purchased, the object product must be purchased. ■ **Compatibility - Excludes.** If the subject product is purchased, the object product must not be purchased.
Scope	You can configure the product so this field can be used to select the scope of the rule, the set of products that the rule applies to. It displays the option: ■ **Projected Asset Cache.** If you are using asset-based ordering, select this to apply the rule to the Account Projected Assets. For more information about projected asset cache, see *Siebel Order Management Infrastructure Guide*. You can add other options by configuring the product using Oracle's Web Tools. In addition, the application does not evaluate the scope field unless you configure the product using Oracle's Web Tools. You must select Column Displayed from the menu to display this field.
Object Product	Select the object product. **NOTE:** Make an entry in only one of the fields Object Product, Object Product Line, Object Product Class.
Object Product Line	Select the object product line.
Object Product Class	Select the object product class.
Start Date	Enter the date when the rule goes into effect.
End Date	Enter the date when the rule is no longer in effect.
Matrix Name	Select the compatibility group used for grouping this rule. For more information about compatibility groups, see "Defining Compatibility Groups" on page 372.

5 If you are creating a compatibility rule that has the current product as object, add a new record to the Object Compatibility Rules list and complete the required fields. These are similar to the fields used for defining subject compatibility rules, except that for the fields described in the following table.

Field	Comments
Subject Product	Select the subject product. **NOTE:** Make an entry in only one of the fields Subject Product, Subject Product Line, Subject Product Class.
Subject Product Line	Select the subject product line.
Subject Product Class	Select the subject product class.
Object Product	Select the object product.

6 Click Clear Cache.

Defining Compatibility Rules for Product Lines

Before you can define a product line compatibility rule, you must define the product lines, and if necessary the products and product classes, it applies to, as described in *Siebel Product Administration Guide*.

To define compatibility rules for a product lines

1 Navigate to the Administration - Product screen.

2 Click the Eligibility and Compatibility Rules view tab.

3 In the Eligibility and Compatibility Rules link bar, click Product Line Compatibility.

4 Add a record to the Subject Compatibility Rules list or the Object Compatibility Rules list, and complete the necessary fields.

These fields are similar to the fields used for product compatibility, except that you must choose a product line instead of a product as the subject product of a subject compatibility rule and as the object product of an object compatibility rule.

Defining Compatibility Rules for Promotions

Before you can define promotion compatibility rule, you must define the product promotions it applies to, as described in the topic on product promotions in *Siebel Pricing Administration Guide*.

To define promotion compatibility rules

1 Navigate to the Administration - Product screen.

2 In the link bar, select Product Promotions.

3 In the Product Promotions list, select the promotion you are defining rules for.

4 Click the Eligibility and Compatibility Rules view tab.

5 In the link bar of the Eligibility and Compatibility Rules view, click Product Compatibility.

6 Add records to the Subject Compatibility Rules list and Object Compatibility Rules list, and define the rules in the same way that you do when you are defining product compatibility rules.

Creating Compatibility Matrices

When you create a compatibility matrices, you enter data similar to the data entered in a compatibility rule, but you enter it directly under the matrix record.

You must use compatibility matrices to create compatibility rules where the subject is a product class.

Eligibility and compatibility rules are shown in the Eligibility and Compatibility Matrices view, regardless of how you created them. If you create them in Eligibility and Compatibility Rules view and grouped them under a matrix, they are displayed under this matrix. This procedure describes how to create them directly under the matrix.

To create a compatibility matrix

1 Navigate to the Administration - Product screen, then the Eligibility and Compatibility Matrices view.

2 In the Eligibility and Compatibility Matrices list, add a new record and In the Eligibility and Compatibility Matrices list, add a new record and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for a group.
Matrix Type	Select Compatibility Rules.

3 Click the name of the new record.

The Compatibility Matrix form and the Compatibility Rules list appear.

4 Add a new record to the Compatibility Rules list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Subject Product	Select the subject product. The default is the product that is selected in the Products list, but you can select any product to replace it. **NOTE:** Make an entry in only one of the fields Subject Product, Subject Product Line, and Subject Product Class.
Subject Product Line	Select the subject product. The default is the product that is selected in the Products list, but you can select any product to replace it.
Subject Product Class	Select the subject product. The default is the product that is selected in the Products list, but you can select any product to replace it.
Type	Select the type of the rule. Options are: ■ **Compatibility - Requires.** If the subject product is purchased, the object product must be purchased. ■ **Compatibility - Excludes.** If the subject product is purchased, the object product must not be purchased.
Object Product	Select the object product. **NOTE:** Make an entry in only one of the fields Object Product, Object Product Line, Object Product Class.
Object Product Line	Select the object product line.
Object Product Class	Select the object product class.
Effective Start	Enter the date when the rule goes into effect.
Effective End	Enter the date when the rule is no longer in effect.

Verifying Quotes and Orders for Eligibility and Compatibility

You can verify the eligibility and compatibility for all the products in a quote or order or for individual line items.

It is useful to verify all products before a quote changes status or is taken to a different stage of the order management process. This is especially important for products that have compatibility rules written against them.

When a product is added to a quote or order, eligibility and compatibility is evaluated only for the added line item. Thus, if the added item is incompatible with a product already in a quote, only the added item will show this, not the item already on the quote.

For example, a compatibility-excludes rule says that product A and product B are incompatible. The user adds product A to the quote, and its eligibility status is marked as Y. Then the user adds product B to the quote, and B's eligibility status is marked as N, but A's status is still Y. When you click the Verify button, all items on the quote are evaluated, so product A's eligibility status also becomes N.

To verify eligibility and compatibility for all products in a quote or order

1. Navigate to the Quotes or Orders screen, then the List view.
2. In the list, select the desired quote or order.
3. Drill down on the name of the quote or order.
4. Click the Line Items view tab.
5. In the Quote or Order header form, click Verify.

To verify eligibility and compatibility for one product in a quote or order

1. Navigate to the Quotes or Orders screen, then the List view.
2. In the list, select the desired quote or order.
3. Drill down on the name of the quote or order.
4. Click the Line Items view tab.
5. In the Line Items list, select the desired line item.
6. From the Line Items menu, select Verify.

Eligibility and Compatibility Workflow Reference

Eligibility and Compatibility workflows are covered in the following topics:

The PSP Driver Workflow is the default controlling workflow for all these Eligibility and Compatibility Procedures. When the code raises a signal to call the controlling workflow for eligibility and compatibility, it passes the names of these Eligibility and Compatibility Procedures to the PSP Driver Workflow. You can also configure your own controlling workflow to replace this default controlling workflow. For more information about the PSP Driver Workflow, see *Siebel Order Management Infrastructure Guide*.

Eligibility and Compatibility Procedures are PSP procedures. For more information, see the topic about PSP Procedures in *Siebel Order Management Infrastructure Guide*.

For more information about workflows, see *Siebel Business Process Framework: Workflow Guide.*

Product Eligibility & Compatibility - Default Workflow

Product Eligibility & Compatibility - Default workflow is responsible for determining the eligibility and availability of a list of inputted row set of Products. Eligibility and availability are based on eligibility rules setup in the administration views and in the procedure itself. The workflow takes a row set of Products and flags each row with an eligibility status and also a comment detailing the reason for ineligibility, if any.

This workflow is shown in Figure 15.

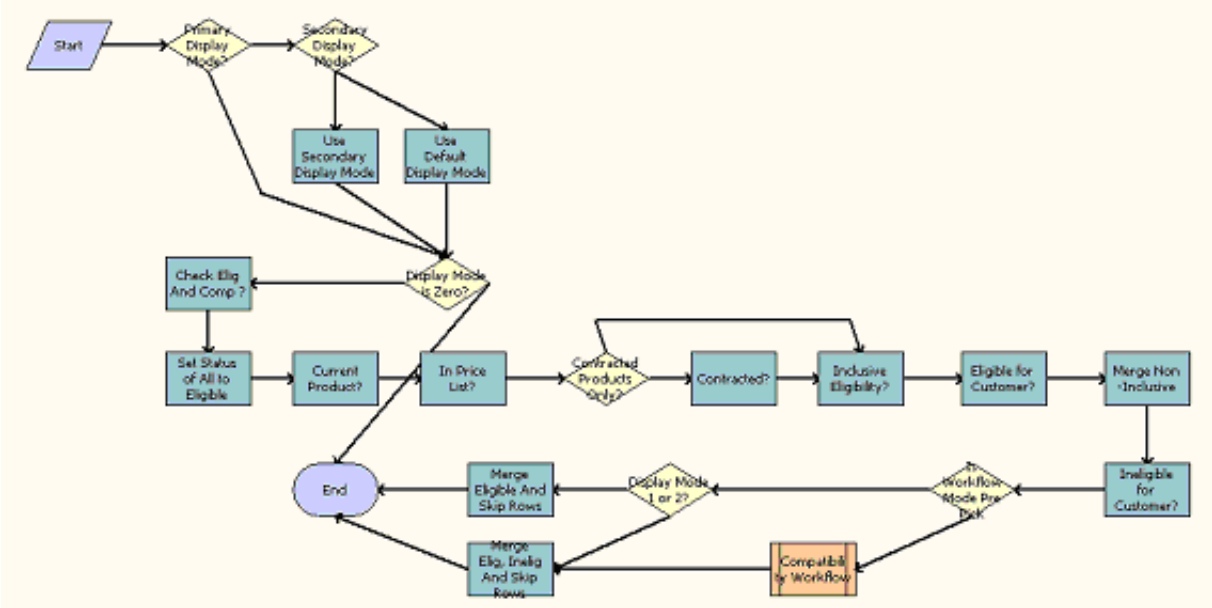

Figure 15. Product Eligibility & Compatibility - Default Workflow

Workflow Description. This workflow does the following:

- **Primary Display Mode?.** Verifies whether Primary Display Mode is specified.
- **Secondary Display Mode?.** Verifies whether Secondary Display Mode is specified.
- **Use Secondary Display Mode.** Assigns the Secondary Display Mode to the Display Mode variable.
- **Use Default Display Mode.** Sets Default Display Mode = 1 if both the Primary and Secondary Display Modes are not specified.

- **Display Mode is Zero?.** Verifies whether if Display Mode is zero and branches to the End of procedure if zero.
- **Set Status of All to Eligible.** Sets each Row's eligibility status to Yes. Defaults all products to eligible. Subsequent steps will flag ineligible products.
- **Current Product.** Verifies dates associated with the current line item. If the line item is outside of the effective dates, then sets the eligibility status to No and moves the product into the Output Set. Ensures that each product can be sold today by checking its effective from and effective to dates.
- **In Price List.** Searches the Price Line Item Buscomp for the product and if the product does not exist, sets eligibility status to No. Ensures that each product is in the current price list.
- **Contracted Product Only.** Decides whether it needs to check for contract eligibility. Checks against contracted products if the account Contracted Product Only Flag = Y.
- **Contracted?.** Queries for the Context Account id under Agreement BC, and verifies the start and end dates. Set eligibility status to No if no valid contracts are found. Ensures that each product appears in a pricing contract for the account.
- **Inclusive Eligibility.** Splits the Row Set based on the Inclusive Eligibility Flag. Only checks inclusive eligibility rules for products whose Inclusive Eligibility Flag = Y.
- **Eligible for Customer.** Queries the Product Eligibility BC and verifies that the row set items satisfies the eligibility requirements (Inclusive). Ensures that each product satisfies an inclusive eligibility rule (for example, that it is sold in the customer's state).
- **Merge Non-Inclusive.** Merges the Non-Inclusive Eligibility Row Set into the Row Set.
- **InEligible for Customer.** Queries the Product Eligibility BC and verifies that the row set items satisfies the eligibility requirements (non-inclusive). Ensures that no ineligibility rules are broken for each product (for example, that the product is not sold in customer's postal code).
- **Is Workflow Mode Pre-Pick?.** Verifies whether Workflow Mode is Pre-pick and branches to Compatibility workflow if not Pre-Pick.
- **Compatibility Workflow.** Executes the Product Compatibility - Default Workflow to check for compatibility of Products in Row-Set.
- **Display Mode 1 or 2?.** Verifies the value of the Display Mode property.
- **Merge Eligible and Skip Rows.** Merges Eligible and Skip Rows for Mode = 2.
- **Merge Elig, Inelig and Skip Rows.** Merges Eligible, Ineligible and Skip Rows for Mode = 1.

Product Compatibility - Default Workflow

Product Compatibility - Default workflow is responsible for determining other required and excluded products for a list of inputted row set of products. It determines required and excluded products based on compatibility rules set up in the administration views. The workflow takes a row set of Products and flags each row with compatibility status and with a comment detailing related products that are either required or excluded.

This workflow is shown in Figure 16.

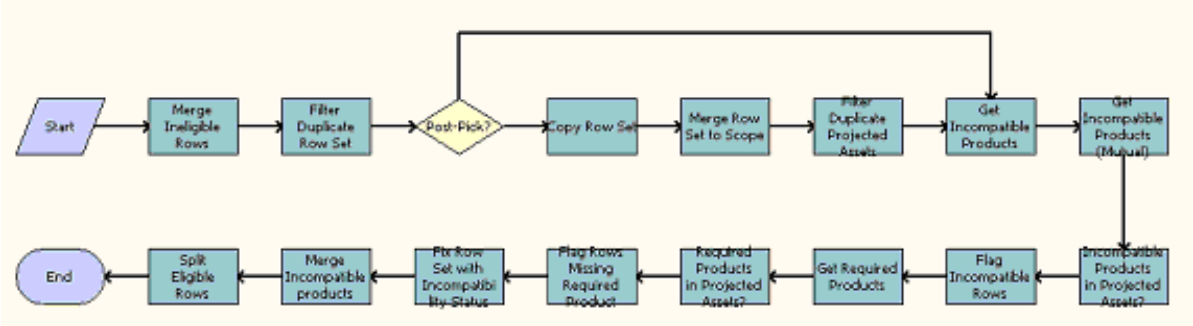

Figure 16. Product Compatibility - Default Workflow

Workflow Description. This workflow does the following:

- **Merge Ineligible Rows.** Merges the row sets passed in by parent workflow Product Eligibility & Compatibility - Default, which already determined Eligibility of the products and separated the products into row sets depending on whether or not the products are eligible
- **Filter Duplicate Row Set.** Filters out any duplicate products in the row set.
- **Post-Pick?.** Determines if the workflow is run in post-pick or pre-pick mode. Skips steps Copy Row Set, Merge Row Set to Scope and Filter Duplicate Projected Assets if in not in Post-Pick mode.
- **Copy Row Set.** Makes a copy of the unique row set created in the step Filter Duplicate Row Set.
- **Merge Row Set to Scope.** Merges row sets to form the scope row sets. Scope row set is all the products that the input row set will be checking Compatibility rules against. It includes the Projected Assets as well as products in the Customizable Product.
- **Filter Duplicate Projected Assets.** Filters any duplicate products in the Projected Assets row set.
- **Get Incompatible Products.** Loads A Excludes B rules for the products from the Compatibility administration view.
- **Get Incompatible Products (Mutual).** Load B Excludes A rules for the products.
- **Incompatible Products in Projected Assets?.** Checks whether there are incompatible products in the Project Asset row set based on rules retrieved in the two previous steps.
- **Flag Incompatible Rows** Flags rows in the Row Set with Status and Comment for incompatibilities found in the previous step.
- **Get Required Products.** Loads A Requires B rules for the products.
- **Required Products in Projected Assets?.** Checks if there are required product rule violated for products in the Project Asset row set based on rules retrieved in the previous steps.
- **Flag Rows Missing Required Product.** Flags rows in the Row Set with Status and Comment for missing products based on violations found in the previous step. Moves all rows with violations to the Incompatible Row Set

- **Fix Row Set with Incompatibility Status.** Removes from the Row Set all the rows that exist in the Incompatible Row Set.
- **Merge Incompatible products.** Merges the Incompatible Row Set with the Row Set.
- **Split Eligible Rows.** Splits the rows with Compatibility violations from the Row Set to the Ineligible Row Set. Ineligible Row Set now contains a row set of all the products that have Eligibility violations or Compatibility violations.

Compatibility Multiple Popup Workflow

Compatibility Multiple Popup Workflow is responsible for displaying product compatibility violations determined by the Eligibility and Compatibility workflows. This popup gives the user the option of adding the incompatible products or removing them. The popup gets triggered in Quotes and Orders unless the Skip Quote flag is set to N.

This workflow is shown in Figure 17.

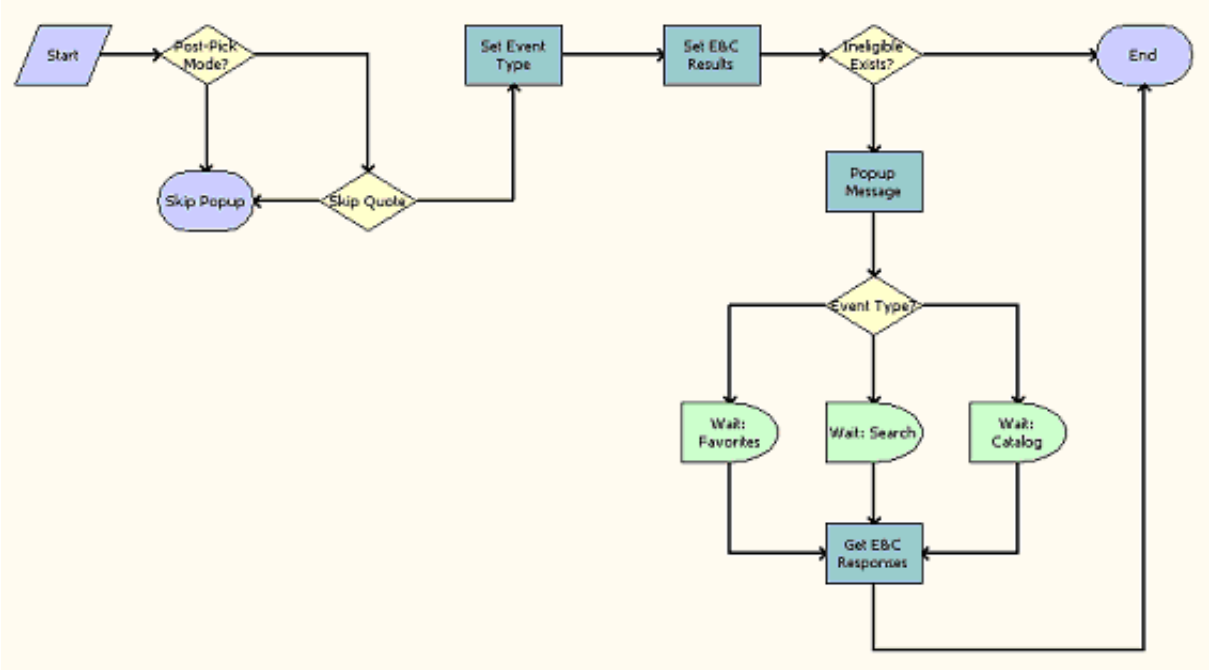

Figure 17. Compatibility Multiple Popup Workflow

Workflow Description. This workflow does the following:

- **Post-Pick Mode?.** Skips this workflow completely if in Pre-pick mode.
- **Skip Quote.** Ends this workflow if the Skip Quote flag is Y.
- **Set Even Type.** Stores the full event name after reading the event prefix and the event type. This event name is used later to resume the workflow.

- **Ineligible Exists?.** If there are no Eligibility and Compatibility (E&C) violations, goes to end of workflow
- **Popup Message.** Displays the popup with violating product and comment and waits for the user to decide whether to continue to add this product or to remove this product.
- **Event Type?.** Checks if the event for this workflow comes from Favorites, Search, or Catalog
- **Wait Favorites/Search/Catalog.** Waits for a resume workflow event type from one of the 3 modules.
- **Loop**
- **Get E&C Responses.** Depending on user input from previous steps, deletes or keeps products.

Configurator Eligibility Compatibility Workflow

This Workflow is invoked by Siebel Configurator to check if there are any products or attributes that violate Eligibility or Compatibility rules. There are 2 types of checking: Pre-Pick check on the domains of products and attributes in Siebel Configurator, and Post-Pick check on the products and attributes that are selected in the Siebel Configurator Instance.

This workflow is shown in Figure 18.

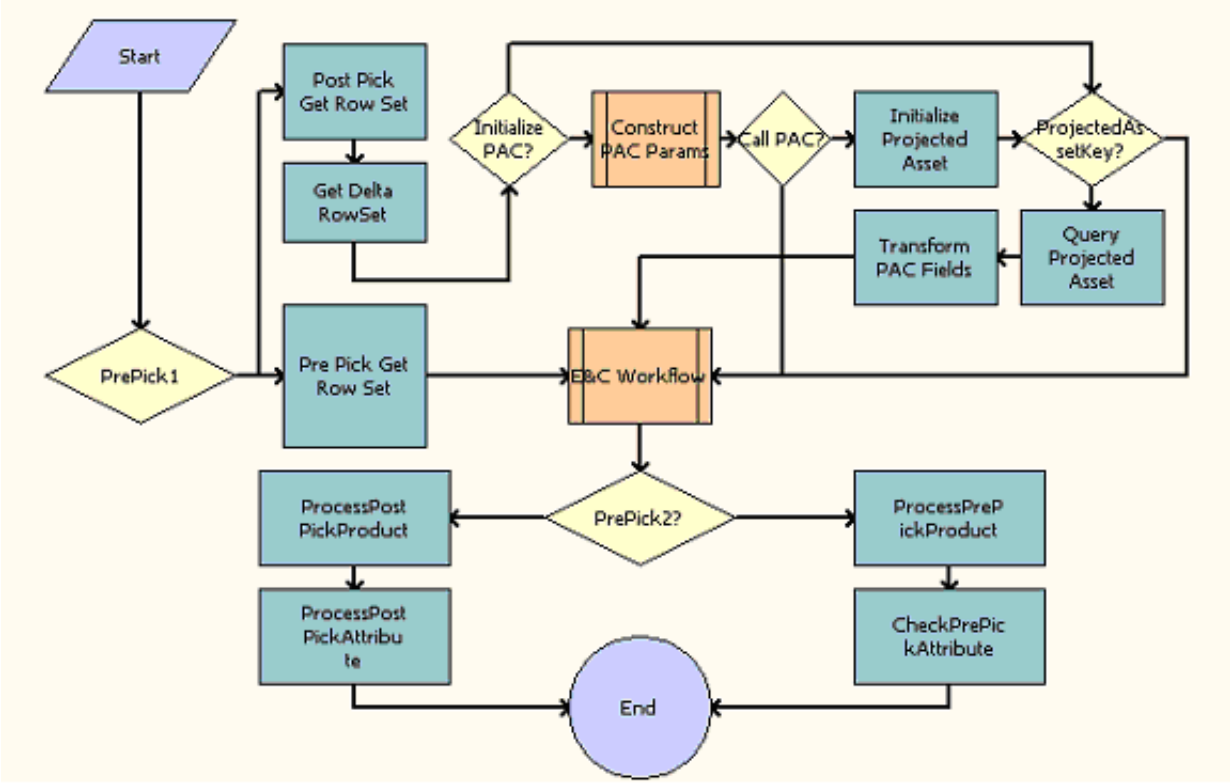

Figure 18. Configurator Eligibility Compatibility Workflow

Workflow Description. This workflow does the following:

- **PrePick1.** Determines whether this is a Pre-Pick or Post-Pick scenario.
- **Pre Pick Get Row Set.** Creates a Row Set of all the Domain Products in Siebel Configurator.
- **Post Pick Get Row Set.** Asks the Context Service to generate a Row Set of the current Siebel Configurator Instance.
- **Get Delta RowSet.** Generates a subset of the Row Set generated in the previous step. This subset contains only the newly selected or deleted instance after a Siebel Configurator Submit Request. The Compatibility Engine will use this Row Set and compare it with the Row Set generated in the previous step to determine if any Compatibility Rules are violated.
- **Initialize PAC?.** Determines whether Projected Asset Cache (PAC) was already initialized.
- **Construct PAC Params.** Constructs the necessary input arguments that PAC business service needs. This will control which buscomps (Quote, Order, or Asset) that will be queried by PAC.
- **Call PAC?.** Based on the inputs set in the previous step, determines whether the application needs to call PAC.
- **Initialize Projected Asset.** Creates the Projected Asset Cache.
- **Projected Asset Key.** Creates a unique key for the Cache created in the previous step.
- **Query Projected Asset.** Creates a Property Set of the items cached in the PAC.
- **Transform PAC Fields.** Transforms the PAC fields to Variable Names that are recognized by the Eligibility/Compatibility engine
- **E&C Workflow.** Passes the control to the Product Eligibility & Compatibility - Default workflow.
- **PrePick2.** Determines whether this is a Pre-Pick or Post-Pick scenario.
- **ProcessPostPickProduct.** Updates Siebel Configurator instances that are Ineligible.
- **ProcessPostPickAttribute.** Determines the eligibility of attributes that are already selected (part of the Instance).
- **ProcessPrePickProduct.** Updates Siebel Configurator Domain Products that are Ineligible.
- **CheckPrePickAttribute.** Determines the eligibility of the domains of the attributes.

Check Eligibility & Compatibility - Default Workflow

This workflow is responsible for checking the eligibility and compatibility of a quote or order line item.

This workflow is shown in Figure 19.

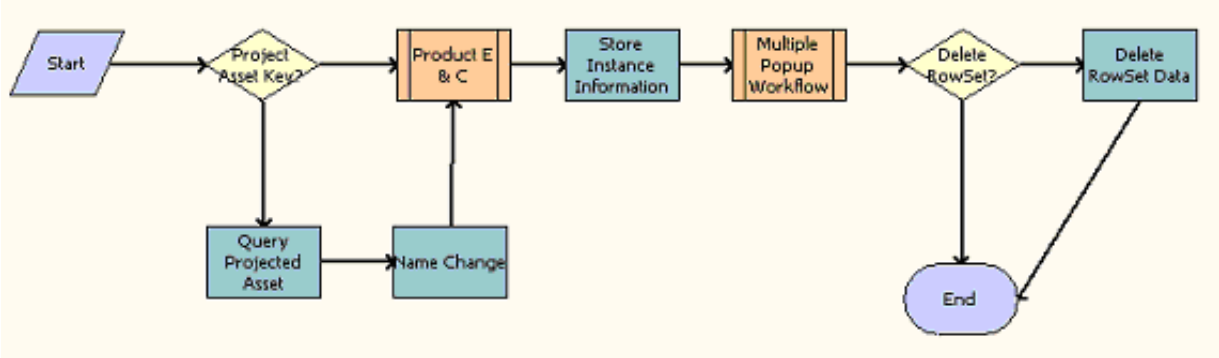

Figure 19. Check Eligibility & Compatibility - Default Workflow

Workflow Description. This workflow does the following:

- **Project Asset Key?**. Determines whether the Projected Asset Key is present.
- **Query Projected Asset.** If the Projected Asset Key is present, searches for the asset.
- **Name Change.** Calls the Conditional Action Transform for the Business Service Row Set Transformation Toolkit.
- **Product E & C.** Calls the Product Eligibility & Compatibility - Default workflow.
- **Store Instance Information.** Stores instance information that will be used in the Delete RowSet Data step.
- **Multiple Popup Workflow.** Calls the Compatibility Multiple Popup Workflow.
- **Delete RowSet?.** Determines whether any Line Items in the Row Set have been marked for deletion.
- **Delete RowSet Data.** Deletes any Line Items that are marked for deletion.

Pricing and Eligibility Procedure - Default Workflow

Pricing and Eligibility Procedure - Default workflow will check the Eligibility and Compatibility of a Line Item and then do Pricing.

This workflow is shown in Figure 20.

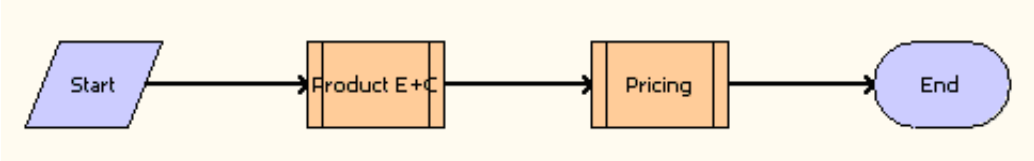

Figure 20. Pricing and Eligibility Procedure - Default Workflow

Workflow Description. This workflow does the following:

- **Product E + C.** Calls the Check Eligibility & Compatibility - Default workflow, which checks eligibility and compatibility for the Line Items.
- **Pricing.** Calls the Pricing Procedure - Default workflow, which prices the line items.

23 Creating Validation Rules for Customizable Products

This chapter covers product validation. It includes the following topics:

About Validation for Customizable Products

The Product Validation view:

- Allows you to validate configurable products using the provided Simple Expression business service.
- Provides the infrastructure for validating products using custom validation business services that that you create.

The product ships with one validation business service, named VORD CPVE Simple Validation, which validates the components that make up a customizable product by using validation expressions.

The Product Validation service can be run in the following ways:

- When the user selects Verify from the Line Item menu of the Quote or Order screens.
- When the user clicks the Verify button within the Siebel Configurator user interface.
- When the user selects Save or Done.

When the user verifies from the Quote or Order screen, the Product Validation service is run in conjunction with other verification services. The messages it generates appear with the messages from those services.

The messages from the Product Validation business service do not prevent the user from continuing, past the messages.

The Product Validation provides two key benefits:

- It provides the architecture to create custom validation business services that are processed in conjunction with all other customizable product behavior. Depending on how the custom service is written, it could use rules written in an administration applet, or it may have the validation logic hard-coded into the script.

- It fires only when called by clicking the Done, Save, or Verify button, so it makes batch validation possible in a configuration session. Product validation rules are not fired every time the selection changes, as configuration constraints are.

NOTE: Product validation uses the Projected Asset Cache, which contains the customer's current assets, current open order lines and the line items from the current quote. For more information about the Projected Asset Cache, see *Siebel Order Management Infrastructure Guide*.

Scenario for Product Validation Using Custom Validation Services

This scenario describes one hypothetical way that product validation can be used. You may use it differently, depending on your business model.

An automotive wholesaler buys cars with many factory-installed options and configures them by adding wholesaler-installed options.

Some factory-installed options are not compatible with some wholesaler-installed options. The wholesaler must create requires and excludes rules such as:

- Wholesaler-installed rear spoiler requires factory-installed V-6 engine.
- Factory-installed sunroof excludes wholesaler-installed roof rack.

These are similar to the requires and excludes rules defined in Siebel Configurator. However, the wholesaler cannot use Siebel Configurator to configure these products, because the wholesaler only controls some of the components in the configuration.

For example, if you created a rule in Siebel Configurator saying that sunroof excludes roof-rack, Siebel Configurator would remove the sunroof when a user adds a roof rack. However, the wholesaler does not have control of the sunroof, and needs a rule saying that the roof rack cannot be installed if there is a sunroof.

To solve this problem using product validation, the wholesaler:

- Uses the Pricing Administration screen, then the Attribute Adjustments view to create dynamic matrix tables of allowable combinations, as shown in Table 44 and Table 45.

Table 44. Allowable Combinations of Rear Spoiler and Engine

Rear Spoiler	Engine Type
Y	6
Y	6T
N	*

Table 45. Allowable Combinations of Sunroof and Roof Rack

Sunroof	Roof Rack
Y	N
N	Y
N	N

- Write a validation business service for each of these matrices. The services reference the matrix to determine whether a combination that the end user selects is valid.
- In the Product Validation view, add these business services and add the messages that are displayed to the user if the rules are violated. There is no limit to the number of business services that you can add for one validation.

Activating Workflows for Product Validation

Product validation is based on Siebel Workflows. You must activate these workflows before using the feature. For information about activating workflows, see *Siebel Business Process Framework: Workflow Guide*.

Activate the following workflows:

- VORD Validate (Order)
- VORD Validate (Quote)

Setting Up Product Validation Using the Simple Expression Business Service

To perform product validation using the simple expression business service, which is provided with the product, perform the following tasks:

1. "Creating the Customizable Product for Validation" on page 390
2. "Creating the Messages for Product Validation" on page 390
3. "Adding the Validation Services Record" on page 391
4. "Creating Product Validation Expression Rules" on page 391

Creating the Customizable Product for Validation

Validation rules only work on products that have been defined as components of a product with components. They use the Instance ID of the product with components as the key to retrieve products to validate.

As the first step in creating validation rules, define a product with components whose components include all the products that the rules will apply to. For more information, see Chapter 9, "Designing Products with Components."

Creating the Messages for Product Validation

You must use the Administration - Application screen, then the Message Types view to define the error messages that the product validation rule will display.

For more information about defining messages, see *Siebel Order Management Infrastructure Guide*.

To create the messages for product validation

1. Navigate to the Administration - Order Management screen, then the Message Types view.
2. In the All Message Types list, add a new record for each message and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the message record. When you add rules in the Simple Validation Expression Rules list, this name will be used to link the rule to a message.
Group	If you are using the validation service VORD CPVE Simple Validation, which ships with the product, as the group, you must enter *Simple Expression Rule*.
Full Text	Enter the error message that is displayed by the application. **NOTE:** Only the Full Text field is used by the product validation engine. The Short Text field is ignored.

Adding the Validation Services Record

The Validation Services record contains the name of the validation business service to call.

This task is a step in "Setting Up Product Validation Using the Simple Expression Business Service" on page 390.

To add the Validation Services record

1. Navigate to the Administration - Product screen, then the Products view.
2. Click the Product Validation view tab.
3. In the link bar of Product Validation view, click Validation Services.
4. Add one or more new records to the Validation Services list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Sequence	Enter a number that controls the order in which the validation services are executed. You can create an unlimited number of validation services.
Business Service	Select the business service used to execute this rule. In this case, select VORD CPVE Simple Validation, the provided product validation business service that ships with the product.
Rule Type	Select a rule type. The options are as follows: ■ **Compound.** This business service runs in the context of the Network Validation routines. For information about network validation, see *Siebel Order Management Guide Addendum for Industry Applications*. ■ **Complex.** This business service runs in the context of the Product Validation routines. ■ **All.** This business service runs in both contexts. In this case, you must select Complex or All for use with the VORD CPVE Simple Validation business service.

Creating Product Validation Expression Rules

The Product Validation Expression list allows you to create rules based on expressions that are true or false. If the expression is true, the application displays the error message that you select in the Message field.

This list allows you to create a number of different types of expressions. You may create only one type of expression in each rule.

For example, you can use the Search Expression field to create an expression in Siebel Query Language, such as *[Quantity] > 2*. If the value in the Quantity field is greater than 2, this expression evaluates as true, and the error message is displayed.

To create product validation rules

1. Navigate to the Administration - Product screen, then the Products view.
2. Click the Product Validation view tab.
3. In the link bar of Product Validation view, click Product Validation Expression Rules.

4 Add new records to the Product Validation Expression Rules list and complete the necessary fields.

Table 46 includes some sample simple expression rules that you could use to validate network ordering. The first six columns contain the values you enter in each record, and the last column contains an explanation of what this rule does. For more information about network ordering, see *Siebel Order Management Guide Addendum for Industry Applications*.

Table 46. Samples of Simple Expression Rules

Seq	Error Text	Search Expression	Aggregate Function	Group By Fields	Having Expression	Explanation
1	[Count] [Product]s are missing Service Addresses	([Network Element Type] = "Network Node" AND [Service Address] = "")	Count	[Product Name]	[Count]>0	Validates that all Nodes have a Service Address.
2	[Count] [Product] are missing a "Service Address" and/or a "To Service Address"	([Network Element Type] = "Network Connection" AND (([Service Address] = "") OR ([To Service Address] = "")))	Count	[Product Name]	[Count]>0	Validates that all Connections have a Service Address associated with each end of the connection.
3	[Product] at [Service Address] must have a different "To Service Address"	([Network Element Type] = "Network Connection" AND ([Service Address] = [To Service Address]))				Validates that the Addresses for each end of a connection are different.
4	[Count] [Product] are missing a From Node and/ or To Node	([Network Element Type] = "Network Connection" AND ([Node] = "" OR [To Node] = ""))	Count	[Product Name]	[Count]>0	Validates that all connections have a node associated with each end of the connection.

In addition to the fields described in Table 46, use the Message field to select a message associated with the expression.

After this is setup, whenever a customizable product is validated, for each item in that current instance of the product, each expression rule is processed once. If any expression matches the current row, then the message associated with that rule is displayed.

Setting Up Product Validation Using Custom Validation Services

To create product validation rules, perform the following tasks:

1. "Creating the Customizable Product for Validation" on page 394
2. "Creating Messages for Product Validation" on page 394
3. "Creating a Custom Business Service for Product Validation" on page 395
4. "Adding the Validation Services Record" on page 395

Creating the Customizable Product for Validation

Validation rules only work on products that have been defined as components of a product with components. They use the Instance ID of the customizable product as the key to retrieve products to validate.

As the first step in creating validation rules, define a product with components whose components include all the products that the rules will apply to. For more information, see Chapter 9, "Designing Products with Components."

Creating Messages for Product Validation

If your custom business service uses Universal Messaging Service, you must use the Administration - Application screen, then the Message Types view to define the error messages that the product validation rule will display.

For more information about defining messages, see *Siebel Order Management Infrastructure Guide*.

To create the messages for product validation

1. Navigate to the Administration - Application screen, then the Message Types view.
2. In the All Message Types list, add a new record for each message and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Name	Enter a name for the message record. When you add rules in the Simple Validation Expression Rules list, this name will be used to link the rule to a message.

Field	Comments
Group	If you are writing a custom validation service, give the group the name that you will call in the validation service.
Full Text	Enter the error message that is displayed by the application. **NOTE:** Only the Full Text field is used by the product validation engine. The Short Text field is ignored.

Creating a Custom Business Service for Product Validation

You can create your own business services to solve specialized business problems, such as the problem described in "Scenario for Product Validation Using Custom Validation Services" on page 388.

This business service must follow the guidelines described in "About Creating Custom Rule Checkers" on page 396.

For more information about creating business services, see *Siebel eScript Language Reference* and *Siebel VB Language Reference*.

Adding the Validation Services Record

The Validation Services record contains the name of the validation business service to call. In this case, you use it to call the custom business service that you created.

To create validation services

1. Navigate to the Administration - Product screen, then the Products view.
2. Click the Product Validation view tab.
3. In the link bar of Product Validation view, click Validation Services.
4. Add one or more new records to the Validation Services list and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Sequence	Enter a number that controls the order in which the validation services are executed. You can create an unlimited number of validation services.

Field	Comments
Business Service	Select the custom business service that you created to execute this rule.
Rule Type	Select a rule type. Options are: ■ **Compound.** This business service will run in the context of the Network Validation routines. For information about network validation, see *Siebel Order Management Guide Addendum for Industry Applications*. ■ **Complex.** This business service will run in the context of the Product Validation routines. ■ **All.** this business service will run in both contexts. The option you select depends on the custom validation service.

About Creating Custom Rule Checkers

The Compound Product Validation Engine business service invokes a number of different rules checker business services. Two rules checkers are provided with the product, but a customer can build custom rules checker business services that comply with the following API specification.

The following rule checkers are provided with the product:

- **PreValidate.** Returns the list of field names and attribute names used by the rules checker. The list of required fields and attributes may be influenced by the Parameters passed to the rule. For more information, see "PreValidate Method" on page 396.
- **Validate.** Implements the logic of the specific rule and validates the contents of the Projected Asset cache. It returns rules violations. For more information, see "Validate Method" on page 399.

PreValidate Method

The PreValidate method determines the list of fields and attributes that the rules checker requires and returns them to the Compound Product Validation Engine. It may optionally use the product Id to retrieve product specific data related to the rule, or other parameters that may influence the list of fields and attributes.

Example

The Port Over-Subscription Checker business service checks that the sum of the bandwidths of the connections going into or out of a node does not exceed the bandwidth of the node. The checker has two parameters that specify the name of the Bandwidth attribute of the Node product and the name of the bandwidth of the Connection product. The rules checker requires the Network Element Type, Product Name, Node and To Node fields and the Bandwidth attributes to evaluate the rule. The method returns a property set of type Field and a property set of type Attribute containing the list of fields and attributes required by the rules checker.

NOTE: This sample code is provided for instructional purposes only. Different code may be needed, depending on how your application is configured. This sample script contains some fields that may not be present in your application, because they are only available in Siebel Industry Applications.

```
function PreValidate (Inputs, Outputs)

{

   // Retrieve input arguments

   var productId = Inputs.GetProperty("Product Id");

   var parameter;


   // Retrieve the rules checker specific parameters

   // These parameter

   for (var i = 0; i < Inputs.GetChildCount(); i++)

   {

      var child = Inputs.GetChild(i);

      switch (child.GetType())

      {

         case 'Parameter':

         parameter = child;

         break;

         default:

         throw "Unknown argument: " + child.GetType();

         break;

      }

   }

   if (parameter == undefined)
```

```
    {
        throw "Missing input argument 'Parameter'";
    }

    var connectionAttrib = parameter.GetProperty("Connection Attribute");
    var nodeAttrib = parameter.GetProperty("Node Attribute");

    // Define the fields used by this rules checker
    var field = TheApplication().NewPropertySet();
    field.SetType("Field");
    field.SetProperty("Network Element Type", "");
    field.SetProperty("Product Name", "");
    field.SetProperty("Node", "");
    field.SetProperty("To Node", "");

    // Define the attributes used by this rules checker
    var attribute = TheApplication().NewPropertySet();
    attribute.SetType("Attribute");
    attribute.SetProperty(connectionAttrib, "");
    attribute.SetProperty(nodeAttrib, "");

    // Return the required fields and attributes
    Outputs.AddChild(attribute);
    Outputs.AddChild(field);
}
```

Related Topic

"Validate Method" on page 399.

Validate Method

The Validate method implements the logic of the rules checker and returns rules violations. It may optionally use the Product Id to retrieve product specific data related to the rule, or other parameters that may influence the logic of the rule. It then queries the Projected Asset Cache using the supplied Asset Cache Key for rule violations and returns an error string for each to the Compound Product Validation Engine.

Example

The Port Over-Subscription Checker business service checks that the sum of the bandwidths of the connections going into or out of a node does not exceed the bandwidth of the node. The checker has two parameters that specify the name of the Bandwidth attribute of the Node product and the name of the bandwidth of the Connection product. The Validate method first queries the Projected Asset Cache using the Asset Cache Key passed as an input argument for all Network Node components and sorts the output by Node name. It then queries the Projected Asset cache for the sum of the bandwidth attribute for all Network Connection components grouped by the Node field. Finally, it queries the Projected Asset cache for the sum of the bandwidth attribute for all Network Connection components grouped by the To Node field. Using the results from the three queries, the Validate method then calculates the sum of the sum of the bandwidths of the connections going into or out of each node and constructs an error message string for each instance where the bandwidth of the node is exceeded. The error strings are returned in the Rule Violation output argument.

NOTE: This sample code is provided for instructional purposes only. Different code may be needed, depending on how your application is configured. This sample script contains some fields that may not be present in your application, because they are only available in Siebel Industry Applications.

```
function Validate (Inputs, Outputs)

{

   // Retrieve input arguments

   var productId = Inputs.GetProperty("Product Id");

   var product = Inputs.GetProperty("Product");

   var assetCacheKey = Inputs.GetProperty("Asset Cache Key");

   var parameter;


   // Retrieve rules checker specific parameters

   for (var i = 0; i < Inputs.GetChildCount(); i++)

   {

      var child = Inputs.GetChild(i);

      switch (child.GetType())

      {
```

```
        case 'Parameter':
        parameter = child;
        break;
        default:
        throw "Unknown argument: " + child.GetType();
        break;
    }
}
if (parameter == undefined)
{
    throw "Missing input argument 'Parameter'";
}

var connectionAttrib = parameter.GetProperty("Connection Attribute");
var nodeAttrib = parameter.GetProperty("Node Attribute");

// Queries the Projected Asset Cache to retrieve a list of nodes sorted by node
// name.
var assetCacheSvc =
    TheApplication().GetService("VORD Projected Asset Cache");
var svcInputs = TheApplication().NewPropertySet();
var svcOutputs = TheApplication().NewPropertySet();

svcInputs.SetProperty("Asset Cache Key", assetCacheKey);
svcInputs.SetProperty("Search Expression",
    "([Network Element Type] = \"Network Node\")");

var sortByField = TheApplication().NewPropertySet();
sortByField.SetType("Sort By Field");
sortByField.SetProperty("Node", "ASC");
```

```
svcInputs.AddChild(sortByField);

assetCacheSvc.InvokeMethod("Query", svcInputs, svcOutputs);

// Retrieves the result from the output of the Query method
var nodePropSet;

for (var i = 0; i < svcOutputs.GetChildCount(); i++)
{
   var child = svcOutputs.GetChild(i);
   switch (child.GetType())
   {
      case 'Result':
         nodePropSet = child;
         break;
      default:
         throw "Unknown argument: " + child.GetType();
         break;
   }
}

if (nodePropSet == undefined)
{
   throw "Missing output argument 'Result'";
}

// Since a single query cannot get the bandwidth of connections into and out of
// a node, the code retrieves this using two queries, one for connections
// going in and one for connections going out. This maximizes the use of high
// performance C++ code in the projected asset cache and minimizes the work
```

```
// done by this script.

// Get the bandwidth going into each node from the projected asset cache
var assetCacheSvc =
   TheApplication().GetService("VORD Projected Asset Cache");

// Set up the inputs to the Query method
var svcInputs = TheApplication().NewPropertySet();
var svcOutputs = TheApplication().NewPropertySet();

svcInputs.SetProperty("Asset Cache Key", assetCacheKey);
svcInputs.SetProperty("Search Expression",
   "([Network Element Type] = \"Network Connection\")");
svcInputs.SetProperty("Aggregate Field", connectionAttrib);
svcInputs.SetProperty("Aggregate Function", "Sum");

var groupByField = TheApplication().NewPropertySet();
groupByField.SetType("Group By Field");
groupByField.SetProperty("Node", "");
svcInputs.AddChild(groupByField);

var sortByField = TheApplication().NewPropertySet();
sortByField.SetType("Sort By Field");
sortByField.SetProperty("Node", "ASC");
svcInputs.AddChild(sortByField);

// Invoke the Projected Asset Cache Query method
assetCacheSvc.InvokeMethod("Query", svcInputs, svcOutputs);

// Get the Query result
```

```
var nodeFromPropSet;
for (var i = 0; i < svcOutputs.GetChildCount(); i++)
{
   var child = svcOutputs.GetChild(i);
   switch (child.GetType())
   {
      case 'Result':
         nodeFromPropSet = child;
         break;
      default:
         throw "Unknown argument: " + child.GetType();
         break;
   }
}

if (nodeFromPropSet == undefined)
{
   throw "Missing output argument 'Result'";
}

// Get the bandwidth going out of each node from the projected asset cache
var assetCacheSvc =
   TheApplication().GetService("VORD Projected Asset Cache");

ar svcInputs = TheApplication().NewPropertySet();
ar svcOutputs = TheApplication().NewPropertySet();

// Set up the inputs to the Query method
svcInputs.SetProperty("Asset Cache Key", assetCacheKey);
svcInputs.SetProperty("Search Expression",
```

```
   "([Network Element Type] = \"Network Connection\")");
svcInputs.SetProperty("Aggregate Field", connectionAttrib);
svcInputs.SetProperty("Aggregate Function", "Sum");

var groupByField = TheApplication().NewPropertySet();
groupByField.SetType("Group By Field");
groupByField.SetProperty("To Node", "");
svcInputs.AddChild(groupByField);

var sortByField = TheApplication().NewPropertySet();
sortByField.SetType("Sort By Field");
sortByField.SetProperty("To Node", "ASC");
svcInputs.AddChild(sortByField);

// Invoke the Projected Asset Cache Query method
assetCacheSvc.InvokeMethod("Query", svcInputs, svcOutputs);

// Get the Query result
var nodeToPropSet;
for (var i = 0; i < svcOutputs.GetChildCount(); i++)
{
   var child = svcOutputs.GetChild(i);
   switch (child.GetType())
   {
      case 'Result':
         nodeToPropSet = child;
         break;
      default:
         throw "Unknown argument: " + child.GetType();
         break;
```

```
    }
}

if (nodeToPropSet == undefined)
{
    throw "Missing output argument 'Result'";
}

// Create a property set for the errors
var ruleViolation = TheApplication().NewPropertySet();
ruleViolation.SetType("Rule Violation");

// Check whether each node is over-loaded
var nodeFromIndex = 0;
var nodeToIndex = 0;
for (var i = 0; i < nodePropSet.GetChildCount(); i++)
{
    // Get details for the current node
    var thisNode = nodePropSet.GetChild(i);
    var thisNodeName = thisNode.GetProperty("Node");
    var thisNodeBandwidth = parseInt(thisNode.GetProperty(nodeAttrib));
    var thisNodeProduct = thisNode.GetProperty("Product Name");

    // Find the current node in the 'total from' property set
    var fromNodeBandwidth = 0;
    if (nodeFromPropSet.GetChild(nodeFromIndex).GetProperty("Node") == thisNodeName)
    {
        fromNodeBandwidth =
        parseInt(nodeFromPropSet.GetChild(nodeFromIndex).GetProperty("Sum"));
        nodeFromIndex++;
```

```
        }

        // Find the current node in the 'total to' property set
        var toNodeBandwidth = 0;
        if (nodeToPropSet.GetChild(nodeToIndex).GetProperty("To Node") == thisNodeName)
        {
            toNodeBandwidth =
                parseInt(nodeToPropSet.GetChild(nodeToIndex).GetProperty("Sum"));
                nodeToIndex++;
        }

        // Raise an error if the bandwidth of the connections exceeds that of the node
        if (thisNodeBandwidth < (fromNodeBandwidth + toNodeBandwidth))
        {
            ruleViolation.SetProperty(thisNodeProduct + " '"
            + thisNodeName + "' is overloaded (" + (fromNodeBandwidth
            + toNodeBandwidth) + " > " + thisNodeBandwidth + ")", "");
        }
    }

    // Return any errors
    Outputs.AddChild(ruleViolation);
}
```

24 Siebel Configurator Technical Reference

This chapter provides technical information of use to server administrators and integrators. It includes the following topics:

In addition, see Chapter 26, "Siebel Configurator API Reference."

Siebel Configurator Architecture

The key components of Siebel Configurator architecture are as follows:

- **Object Manager.** All services that run within a Siebel application are bound by the Object Manager they are running within. The same applies to all caches as well. Therefore services cannot be shared across object managers and neither can cached objects.
- **UI Business Service.** The UI Business service is used by Siebel Configurator to render the UI. The UI business service binds the structure of the customizable product to the Web templates and submits them to the Siebel Web Engine for rendering to the client browser. The UI service is the means by which the user interacts with Siebel Configurator. A unique instance of the UI service is required for each user.
- **Instance Broker.** The Instance Broker is a service that interacts with the UI Business Service. The Instance Broker maintains all the information about the current instance of the customizable product that the user is configuring. The Instance Broker interacts with other services in response to user requests during a configuration session.

- **Object Broker.** The Object Broker is a service that extracts the customizable product definition from the database for use by other Siebel Configurator services.
- **Config Services.** Config Services consists of factories.
- **Factory.** A factory is a service that translates the customizable product definition retrieved by the Object broker into a format the worker can understand.
- **Constraint Engine or Worker.** The Constraint engine is also called the worker or the Siebel Configurator engine. It is a service that computes solutions and enforces all the constraints associated with the configuration. This includes the declarative portion of the customizable product plus constraints added by the user (user picks).

Siebel Configurator Server Deployment

For more information about deploying Siebel Configurator, see *Siebel Deployment Planning Guide*. For more information about Siebel Configurator performance tuning, see *Siebel Performance Tuning Guide*.

Enabling Snapshot Mode

To use SnapShot Mode, you must turn it on by setting a server parameter. When Snapshot Mode is turned on, the Siebel Configurator server runs using cached objects, factories, and workers as much as possible. This improves performance. For more information about enabling Snapshot Mode, see *Siebel Performance Tuning Guide*.

Enabling Auto Match

When a new version of a customizable product is released, Auto Match adjusts the configuration of the product in a quote, asset, or order to reflect the changes. Auto Match is disabled by default.

For Web Client users, you turn Auto Match on by setting its server parameter to TRUE. Table 47 shows the Auto Match server parameter.

Table 47. Server Parameter for Auto Match

Parameter Name	Display Name	Data Type	Default Value	Description
eProdCfgAutoMatchInstance	Product Configurator - auto match quote on reconfigure.	Boolean	FALSE	When set to FALSE, Auto Match is turned off. When set to TRUE, Auto Match is turned on.

For Developer Web Client users (also called mobile client users), add the following entries to the configuration file used to start the application, for example Siebel.cfg.

```
;; This section will be read for mobile clients only

[InfraObjMgr]

eProdCfgAutoMatchInstance=TRUE
```

Specifying Keep Alive Time for Configuration Sessions

By default, product configuration sessions remain active indefinitely. They do not time out.

You can specify how long product configuration sessions remain active by setting the server parameter for Keep Alive Time. This parameter specifies the time in seconds that a session can remain idle before the session is timed out. The default value of -1 means that the session can remain idle indefinitely and will not be timed out. Table 48 shows this server parameter.

Table 48. Server Parameter for Auto Match

Parameter Name	Display Name	Data Type	Default Value	Description
eProdCfgKeepAliveTime	Product Configurator - Keep Alive Time of Idle Session	Integer	-1	The amount of time in seconds that a configuration session can remain inactive before the session is killed.

Enforcing the Field Length for Entering Advanced Rules

The Advanced Rule template allows you to enter a rule containing several thousand characters. However, the database can store rules that contain only up to 900 characters.

You can revise the business component associated with the Advanced Rule template so that you cannot enter more than 900 characters. This business component is used for populating several lists. Revising the business component enforces the 900 character limit on all these lists. Use Oracle's Web Tools to determine the other lists that are affected.

To enforce the field length

1. In Oracle's Web Tools, open a workspace and then navigate to Object Explorer.

   For more information on using workspace dashboards, see *Using Siebel Tools*.

2. Click the Business Component in Object Explorer, and locate the Rule Designer Dummy List VBC business component.

   It is located in the Rule Designer project.

3 Locate the field called 0 (zero) and set the Text Length value to 900.

4 Deliver the updated workspace.

Displaying RAL in the Constraints View

You can revise the Constraints list to add a field that displays the Rule Assembly Language (RAL) translation of your template rules. This is a useful way to learn how to use RAL to write configuration rules.

You must use Oracle's Web Tools to add the field to the Constraints record, and then recompile the siebel.srf file. You must be familiar with creating and modifying applets in Oracle's Web Tools before performing this procedure.

To revise the Constraints view, perform the following tasks:

1 "Locate the Constraints View Applet"

2 "Modify the Constraints View Applet"

Locate the Constraints View Applet

This task selects a target browser and queries for the Cfg SWE Rule Manager Applet.

To locate the Constraints View applet

1 In Oracle's Web Tools, open a workspace and then navigate to Object Explorer.

For more information on using workspace dashboards, see *Using Siebel Tools*.

2 Select View, Toolbars, Configuration Context.

3 In the Target Browser Group drop-down menu, select Target Browser Config.

4 In Available browser groups, select ALL and transfer it to "Selected browser groups for layout editing."

5 Click OK.

6 Click Applet in the Object Explorer.

7 In the Applets list, query for the Cfg SWE Rule Manager Applet.

Modify the Constraints View Applet

This task adds the Rule Spec field to the Cfg SWE Rule Manager Applet.

To modify the Constraints View applet

1 With the SWE Rule Manager Applet highlighted, select Tools, Lock Project.

2 Right click the highlighted applet record and select Edit Web Layout.

3 In the window displaying the layout, right-click and select Preview from the pop-up menu.

4 Click the Template icon and select the Applet List (Base/EditList) template. It is the default.

5 In the Mode drop-down menu, select 3: Edit List.

6 In the Controls/Columns window, click Rule Spec, and move it to the [field] beside the End date in the applet display.

7 Click Save.

8 Click OK to save your changes.

9 Deliver the workspace.

Turning Off Default Instance Creation

When you add a customizable product to a quote, order, or agreement, a default product instance is created. This causes the default items in the customizable product to display as line items. When the user clicks Customize, another instance is created for the configuration session. The default instance is not used.

For large products with components, creating the default instance can significantly increase the time required to add the customizable product to Line Items to the quote or order. To improve performance, you can turn off default instance creation. When you add a customizable product, this causes it to display as a single line item. The default components do not display as line items.

This will not affect performance when the user clicks Customize since this creates a new product instance. Turning off default instance creation applies only to products with components. It does not apply to bundles.

When you turn off default instance creation and the user launches Configurator, a Configurator violation message is displayed. You must also configure the Siebel application to skip this violation message.

To turn off default instance creation

1 In Oracle's Web Tools, open a workspace and then navigate to Object Explorer.

   For more information on using workspace dashboards, see *Using Siebel Tools*.

2 Locate the Quote Item, Order Entry - Line Items or FS Agreement Item business component.

3 Display user properties.

4 Set the Skip Loading Default Cfg Instance user property to Y.

5 Deliver the workspace.

To skip the violation message

1 In Oracle's Web Tools set the User Property Skip Loading Default Cfg Instance of the Business Service SIS OM PMT Service to Y.

2 Edit the Workflow, SIS OM Edit Delta Quote Line Item by adding a new input argument to the workflow step Reconfigure Product Instance and giving it the following values:

- Input Argument = Skip Required Attribute Warning
- Type = Process Property
- Property Name = Skip Required Attribute Warning

3 Make the same changes described in Step 2 to the following workflows:

- SIS OM Edit Service Order Line Item
- SIS OM Edit Complex Asset Workflow

Revising the Default Cardinalities

When you create a relationship in a customizable product, you can specify a minimum, maximum, and default cardinality. If you do not specify cardinalities, the application uses the following defaults:

- Minimum cardinality = 0
- Default cardinality = 0
- Maximum cardinality = 999

If you do not specify cardinalities this means that users are not required to select any items from the relationship and are limited to selecting a maximum of 999 items.

You can change these defaults as needed. For example, you can set the maximum application default cardinality to a number larger than 999.

To revise the default cardinalities

1 In Oracle's Web Tools, locate the Complex Product Structure BusComp.

2 Within the business component, locate the desired field: Default Cardinality, Max Cardinality, or Min Cardinality.

3 Display the user properties for the field.

4 Set the Pre Default Value user property to the desired amount.

The amount must be an integer that is greater than or equal to 0.

Configuring the Object Broker

If you modify the assignment of *Cfg UI Field* properties on the fields of the *Cfg ISS Prod Def* business component, you must do the following:

- After you make the changes, manually clean the file system's ISS_OBrkCache folder, as described in "Delete Contents of ISS_OBrkCache Directory" on page 415.
- Run all installations of Siebel applications that use the same file system from the runtime repository where you made the changes.

Displaying Fields from S_PROD_INT in Selection Pages

You can add fields from S_PROD_INT to selection pages.

To add the fields from the Product Master tables (S_PROD_INT) to selection pages, perform the following steps:

1. Add Fields to the Cfg ISS Prod Def Buscomp. Add the fields to the Cfg ISS Prod Def Buscomp and define user properties. This buscomp is part of the Object Broker and extracts data from S_PROD_INT.
2. Add Controls to the Repository. You can add a new control to the repository to display the name of this new field.
3. Add SWE Code to the Web Template. Add SWE code to the desired Web template. The SWE code retrieves the field from the business component and displays it in the selection pages. Fields display as text boxes.
4. Delete Contents of ISS_OBrkCache Directory. Delete the contents of the ISS_OBrkCache directory. This forces the system to create a new instance of the customizable product containing the fields.

You can display text fields only for product items or for the product root. This means you can insert the SWE code only in the following places:

- For-each loops that iterate on relationship domains or the children of relationship domains. You cannot insert the code in for-each loops that iterate on attributes or on groups.
- At the root level. The template in which you insert the SWE code must not be called from inside a for-each loop in any other Web template.

The procedures in this topic require you to have a thorough knowledge of Oracle's Web Tools. You must also have a thorough understanding of Siebel Configurator Web template structure.

Add Fields to the Cfg ISS Prod Def Buscomp

This procedure adds the fields you want to display to the Object Broker and recompiles the application. This makes the fields available for display.

To add fields to the Cfg ISS Prod Def Buscomp

1. In Oracle's Web Tools, open a workspace and then navigate to Object Explorer.

   For more information on using workspace dashboards, see *Using Siebel Tools*.
2. Locate the Cfg ISS Prod Def Buscomp in Oracle's Web Tools.
3. Add the desired fields from S_PROD_INT to the buscomp.
4. For each field you add, define a user property called Cfg UI Field.
5. Set the user property value to TRUE.
6. Deliver the workspace.

Add Controls to the Repository

You can add a new control in the repository to show the localized field name for a new field.

To add a new control to the repository

1. In Oracle's Web Tools, locate the applet Cfg Cx Runtime Instance Frame (JS HI).
2. Add a new control.

   For example, you can add Name = lblTest, Caption = TestLabel.
3. Use this control in the template.

Add SWE Code to the Web Template

The following example shows the SWE code you would insert in a Web template to retrieve the Part Number field for display:

```
<swe:control id="swe:101Id+4400" CfgUIControl="CfgLabel" CfgHtmlType="CfgLabel"
property="FormattedHtml" CfgFieldName="Part Number"/>
```

The "id" must be that specified in the for-each loop iteratorName, and the increment amount must be unique within the for-each loop.

If you want to display a field name next to the field value, insert an swe:control statement that extracts the field name from the repository. This allows you to support localization. You can insert the swe:control wherever needed in the template. It does not have to be inside a for-each loop. Here is an example of an swe:control tag that extracts the field name for Part Number from the repository. The "id" in the tag must be present but is not used for anything. The lblPartNumber value is the name of the label control in the repository.

For the Open UI interface, insert the following code:

```
<!-- Template Start: eCfgRelationContentsPriceQuantityJS.swt -->

<table border="0" cellpadding="0" cellspacing="3" width="100%">

<swe:for-each id="500" CfgLoopType="Children" startValue="1500" count="Dynamic"
iteratorName="101Id"

CfgFieldList="CfgFieldName:Quantity, CfgUIControl:lblQuantity,
HtmlAttrib_width:80, HtmlAttrib_align:left, Default:Y*

CfgFieldName:Name, CfgUIControl:lblName, HtmlAttrib_width:250,
HtmlAttrib_align:left, Default:Y*

CfgFieldList="CfgFieldName:Part Number, CfgUIControl:lblPartNo, DataSource:Broker,
NeedRefresh:N, HtmlAttrib_align:center,

HtmlAttrib_width:80*

CfgFieldName:RequireMoreChild, Default:Y*

CfgFieldName:List Price, CfgUIControl:lblListPrice, DataType:DTYPE_CURRENCY,
NeedRefresh:N, HtmlAttrib_align:center, HtmlAttrib_width:80*
```

```
CfgFieldName:Current Price, CfgUIControl:lblYourPrice, DataType:DTYPE_CURRENCY,
HtmlAttrib_align:center, HtmlAttrib_width:80*

CfgFieldName:Explanation, CfgUIControl:lblExplanation, HtmlAttrib_width:70,
HtmlAttrib_align:center*

CfgFieldName:Customize, CfgUIControl:lblCustomize, HtmlAttrib_width:70,
HtmlAttrib_align:center"

>
```

To add SWE code to a template

1. Copy the desired template and give it a new filename.
2. Insert the SWE code into the new template.
3. Add the new template to the Pick UI Style dialog box.
4. Select the new template as the UI control for a relationship or an item.

Delete Contents of ISS_OBrkCache Directory

You must delete the contents of this directory. This makes sure that the application loads your changes when generating a customizable product, rather than loading the objects from the cache directory.

To delete the contents of the ISS_OBrkCache directory

1. Locate the Siebel File System directory.

   To see the directory path or system name for the directory, from the Siebel application Help menu, choose Technical Support.
2. In the Siebel File System directory, locate the ISS_OBrkCache directory.
3. Delete all the files in the ISS_OBrkCache directory.

ASIs for Managing Products

Oracle provides Application Service Interfaces (ASIs) which are business services that allow Siebel applications to share information with other applications through integration servers. There are two ASIs for managing products: External Simple Product and Siebel Simple Product. These are briefly described below. For a full technical description of these ASIs, see *Siebel Application Services Interface Reference.*

External Simple Product

This ASI sends information about simple products created in the Siebel application to an external, third-party application. It allows you to create, update, query, and delete a product in the third-party application. This ASI is intended for inclusion in Siebel workflows that automate exporting products. It does not support sending information about products with components or bundles.

The export feature included in the Siebel Product Administration interface exports basic product information to an XML file and is intended for Siebel-to-Siebel transfers of product records. The External Simple Product ASI exports a much larger set of information about the product.

This information sent includes almost all of the fields in the product record.

This ASI receives the following information:

- Confirmation
- Error messages
- Status
- Product ID

Siebel Simple Product

This ASI receives information about simple products created in third-party applications. It allows users to create, update, query, and delete products in the Siebel application. It ASI is intended for use in automated business processes in third-party applications that need to synchronize the Siebel application to external product masters. This ASI does not support receiving information about products with components or bundles. The information accepted by this ASI includes the following:

- Product name
- Product description
- Part number
- Product attributes and attribute values
- Product unit of measure
- Product classification
- Product type
- Substitute product name
- Literature
- Product catalog
- Product category
- Price list

This ASI sends the following information:

- Confirmation
- Error messages
- Status
- Product ID

Auto Match Business Service for Siebel Configurator

Within certain limitations, the Auto Match business service allows Siebel Configurator to automatically match components in a quote, order, or asset with components in the current version of the product model. This capability is important when the product components in a quote, order, or asset were generated from an old version of a product model or when they were populated directly from an external application with different product models.

This can occur in a few different situations. For example:

- After a software upgrade, such as upgrading from Siebel 6.x to Siebel 7.x, it may be necessary or desirable to regroup product components under different relationships in a product with components.
- During an upgrade, if existing product models are converted to newer product models, regrouping of product components may occur as part of the conversion.

In each of these situations, when product components are regrouped under different relationships, there will be conflicts between the new version of the product with components and existing transaction data in quotes, orders, and assets. Depending on how the products with components have changed, it may be possible to use Auto Match to help resolve such conflicts.

Here is a high-level summary of how Auto Match works: Before Siebel Configurator is launched, the order management workflows run Auto Match to ensure that the product instance from the quote, order or asset is compatible with the latest version of the product with components definition. There are three relationship criteria Auto Match validates. It looks for a product that meets any of these criteria:

- Has no component relationship foreign key
- Has an invalid product relationship foreign key
- No longer appears in a valid product relationship

If it finds a product component that fits any of these criteria then the Auto Match business service looks to see whether that product component belongs to a different relationship under the same parent component in the hierarchy. If it finds a valid relationship, the Auto Match business service updates the relationship foreign key of the product component in the in-memory product instance. If no valid relationship is found then the Auto Match business service generates a warning message and deletes the component from the product instance.

For products with attributes, Auto Match business service looks to see whether any required attributes are missing. If so, it returns an error message in the Auto Match report.

Operating System Environment Variables Used with Siebel Configurator

When you are setting up Siebel Configurator, it can be useful to set the operating system environment variables described in Table 49.

Table 49. Operating System Environment Variables Used with Siebel Configurator

Variable Name	Description
CFG_BROKER_FS	Additional file system directory used by the Broker to support multiple file system locations. The value of this variable is appended to the list of possible locations. When remote Siebel Configurator is used, set this variable on both the operating system of the Application Object Manager and the operating system of the remote Siebel Configurator. It is used by the object broker and object administration.
CFG_IGNORE_MISSING_ELEMENTS	Flag to determine whether to ignore the overwritten attributes with the missing base definitions. The possible values are TRUE or FALSE. When remote Siebel Configurator is used, set this variable on the operating system of the remote Siebel Configurator. It is used by the factory.
CFG_PASSWORD	Login password used in remote service. When remote Siebel Configurator is used, set this variable on the operating system of the Application Object Manager. It is used by the remote proxy.
CFG_USERNAME	Login user name used in remote service. When remote Siebel Configurator is used, set this variable on the operating system of the Application Object Manager. It is used by the remote proxy.

Table 49. Operating System Environment Variables Used with Siebel Configurator

Variable Name	Description
PRESERVE_ENGINE_AND_USER_PICKS	Flag to determine whether the application preserves engine and user pick. The possible values are Y or N. When remote Siebel Configurator is used, set this variable on the operating system of the remote Siebel Configurator. It is used by the instance service.
SIEBEL_ENFORCE_REQUIRED_ATTRIBUTES	Flag to determine whether the application enforces the required attributes. The possible values are Y or N. When remote Siebel Configurator is used, set this variable on both the operating system of the Application Object Manager and the operating system of the remote Siebel Configurator. It is used by the engine and the user interface.
SKIP_HIDDEN_AND_NOT_REQUIRED_ATTRIBUTES	Flag to determine whether the application skips the attributes that are hidden and not required. The possible values are Y or N. When remote Siebel Configurator is used, set this variable on the operating system of the remote Siebel Configurator. It is used by the instance service and session.

Table 49. Operating System Environment Variables Used with Siebel Configurator

Variable Name	Description
SEBL_CFG_SEARCH_LIMIT	The number that determines the search limit. Used to limit the counter for the number of failed searches. Set on all application servers. By setting a very high number, you tell the engine not to exit the solve operation. In some cases, if the engine cannot find a solution after a reasonable number of iterations, it displays an error message, but if this variable has a high value, it keeps trying, which can slow the response time to unacceptable levels. When remote Siebel Configurator is used, set this variable on the operating system of the remote Siebel Configurator.
VOD_PROFILER_ON	Flag to determine whether to collect the statistical data on instrumented functions. The possible values are TRUE or FALSE. When remote Siebel Configurator is used, set this variable on both the operating system of the Application Object Manager and the operating system of the remote Siebel Configurator.

25 Configurator Workflow and Method Reference

This chapter covers the Siebel workflows used by Siebel Configurator, and the methods called by these workflows. It includes the following topics:

Siebel Configurator Workflow Reference

Some features of the order management interface are based on Siebel workflows. You can modify these workflows to suit your own business model using Siebel Business Process Designer. For more information, see *Siebel Business Process Framework: Workflow Guide*.

The following topics cover Siebel Configurator workflow:

Configurator Cleanup Workflow

Configurator Cleanup Workflow, shown in Figure 21, releases the memory that was used by complex product data structures when the product was being customized. It is called when the user is done with a configuration session.

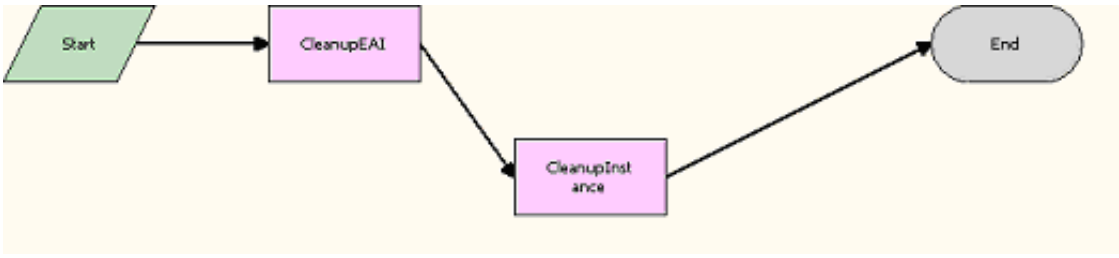

Figure 21. Configurator Cleanup Workflow

Workflow Description. This workflow does the following:

1. **CleanupEAI.** Frees the memory used by the EAI data structure. This step calls CleanupEAI Method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.
2. **CleanupInstance.** Frees the memory used by the CxObj data structure. This step calls CleanupInstance Method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.

Configurator Load Workflow

Configurator Load Workflow, shown in Figure 22, loads the complex product model into memory and displays the Siebel Configurator user interface.

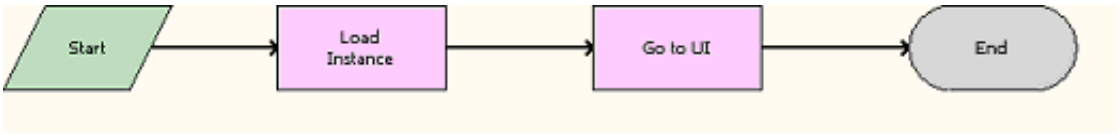

Figure 22. Configurator Load Workflow

Workflow Description. This workflow does the following:

1. **Load Instance.** Loads the Product line item data structure through EAI and creates the CxObj memory structure for it. This step calls LoadEAI Method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.
2. **Go to UI.** Loads the Siebel Configurator user interface for the product.

Configurator Save Workflow

Configurator Save workflow, shown in Figure 23, saves the updated selections for the product that was customized to the line items (database).

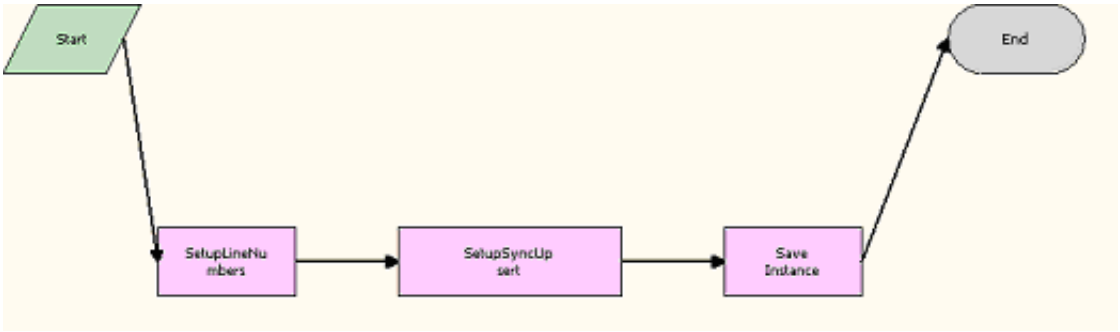

Figure 23. Configurator Save Workflow

Workflow Description. This workflow does the following:

1. **SetupLineNumbers.** Sets up the line numbers for the Line Item being customized corresponding to the CxObj. This step calls SetupLineNumbers Method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.
2. **SetupSyncUpsert.** Updates EAI operation for performance. This step calls SetupSyncUpsert Method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.
3. **Save Instance.** Stores the CxObj to the database using EAI. This step calls StoreEAI Method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.

Configurator Validate Workflow

Configurator Validate Workflow, shown in Figure 24, is used to validate a product model during product administration.

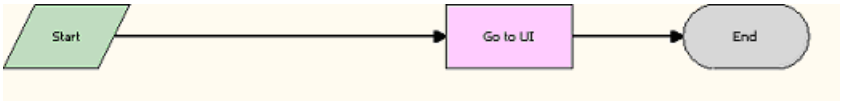

Figure 24. Configurator Validate Workflow

Workflow Description. This workflow does the following:

Go to UI. Loads the Customize UI for the product. This step calls "LoadInstance Method" on page 424.

Configurator External Validate Workflow

Configurator External Validate Workflow, shown in Figure 25, is responsible for releasing the memory used by complex product data structures when the product was being customized. It is called when the user is done with a Customize session.

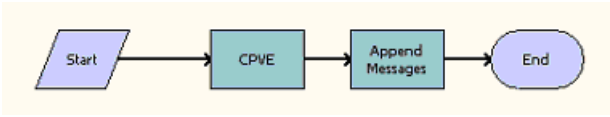

Figure 25. Configurator External Validate Workflow

Workflow Description. This workflow does the following:

1. **CPVE.** Frees the memory used by the EAI data structure. This step calls "Validate Complex Product From Property Set Method" on page 426.
2. **Append Messages.** Frees the memory used by the CxObj data structure. This step calls "AppendMessages Method" on page 427.

Siebel Configurator Methods Reference

The following topics cover the methods that are called by the Siebel Configurator Workflows:

Siebel Configurator workflows also use many methods that are described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.

LoadInstance Method

LoadInstance method is invoked on the Cfg UI Service to display Customize UI. It calls LoadInstance on Complex Object Instance Service to initialize the instance service with CxObj created in the LoadEAI step. The instance service also validates the CxObj against the product definition from Cfg Broker. It also does Promotion validation and Eligibility & Compatibility violation checks. The session with the Rule engine evaluator (ILOG) is also initialized if needed. The UI Service gets the updated CxObj representation from Instance Service and displays the UI based on the product model definition.

This method is part of the Cfg Web UI Service Loader business service. It must not be confused with "LoadInstance Method" on page 431, which is part of Complex Object Instance business service.

NOTE: The LoadInstance method cannot be used without first calling the LoadEAI method, which is described in the Copy Service reference in *Siebel Order Management Infrastructure Guide*.

Syntax

LoadInstance <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
AutoSync	Sync the changes to the CxObj to the DB made during LoadInstance (due to product model definition changes and so on)
Business Component Name	Item BC Name
BusObjName	Business Object name
Cfg Type	eConfigurator
ChangeView	If set to Y, the Customize UI view will be displayed
ComplexProdutId	Root Product Id
Currency Code	Currency Code
DISABLEPRICER	If set to Y, pricing workflows are not called

Input Argument	Description
DonotStartNewSession	If set to Y, StartNewSession is not called on Instance Service
Exchange Date	Exchange Date
HeaderBusCompName	Header BC Name
InstanceName	A string indicating the name of the instance
IntegrationObjName	Integration Object for loading the Line Items
Mode	Quote / Order / Agreement / Asset
NewRecord	If set to Y, default instance will be created
Notify List	The list of business components that need to be notified when the Customize session is finished
ParentObjId	Quote Id / Order Id / Agreement Id / Asset Id
PriceListId	Price List associated with this line item or Quote
Refresh List	The list of business components that need to re-executed when the Customize session is finished
ReturnViewName	View that displays when the Customize session is finished
RowId	Root Line Item Id
SearchSpec	Set this parameter to filter out all other hierarchical instances in the child buscomp
SkipCfgEligibilityCheck	If set to Y, the eligibility check will be skipped
SkipLoadingDefaultInstance	If set to Y, the default instance will not be loaded during New Record
Start Date	Date to use for Broker StartNewSession
Sub Line Item Integration Id	Needed when customizing sub line items
Sub Line Item Product Id	Needed when customizing sub line items
TriggerEvent	If set to Y, trigger the Initialize script event after LoadInstance is done
Type	CxInstService
UIOption [optional]	UI option name
ValidateMode	If set to Y, it indicates that a validation is being performed (no sync when done)
ViewName (JS)	View Name of the Siebel Configurator Open UI view
XABusCompName	XA BC Name

Output Arguments

Input Argument	Description
CxObj	The property set corresponding to the CxObj data structure in Instance service.

Usage

LoadInstance method is invoked on the Cfg UI Service to display the Customize UI. It calls LoadInstance on Complex Object Instance Service to initialize the instance service with CxObj created in the LoadEAI step.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Mobile/ Developer Web Client Automation Server

Validate Complex Product From Property Set Method

Validate Complex Product From Property Set method runs validations against a given record set captured in the form of a property set.

This method is part of the VORD CPVE Validation business service.

Syntax

Validate Complex Product From Property Set <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
IdFieldName	Identifier Field Name. For example, Asset Integration Id
RecordSet	Property Set
RootProductId	Id of the Root Complex Product

Output Arguments

Output Argument	Description
Messages	A property set containing the list of messages to display
Integration Object	This integration object name is SIS OM Quote, SIS OM Order, or SIS OM Asset.

Usage

Validate Complex Product From Property Set method runs validations against a given record set captured in the form of a property set.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Mobile/ Developer Web Client Automation Server

AppendMessages Method

AppendMessages displays the messages on the Customize UI.

This method is part of the Siebel Configurator business service.

Syntax

AppendMessages <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
Messages	A property set containing the list of messages to display

Usage

AppendMessages displays the messages on the Customize UI.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Mobile/ Developer Web Client Automation Server

26 Siebel Configurator API Reference

This chapter summarizes the APIs available to Siebel Configurator and focuses on a segment of those APIs.

This chapter includes the following topics:

About Siebel Configurator APIs

This chapter introduces advanced users to Siebel Configurator APIs. It assumes that you know Siebel Configurator and Siebel server architecture. Implementing the APIs described in this topic also requires proficiency in Siebel EAI and Siebel Object Interfaces.

To use these APIs, the user must be familiar with the following:

- Siebel Business Process Designer
- Runtime Events (personalization) if invoked from the UI
- Siebel Object Interfaces
- A Siebel scripting language (Siebel VB or Siebel eScript)
- Recursive programming techniques
- Constraint satisfaction theory
- Underlying behavior of Siebel Configurator
- Siebel product definition data model
- Siebel property set representation of data (creation and transformation)
- EAI Transports and Interfaces

Available Siebel Configurator APIs

Three main groups of APIs are used for accessing Siebel Configurator:

- Group 1: UI
  - CPRUI Service API as Siebel Web Template items.
  - DOM API within the browser inherent in JavaScript and HTML.
- Group 2: Model
  - Scripts that execute in the context of the current session and are implemented as part of the configuration model in the Scripts view.
- Group 3: Instance
  - This API is for using Siebel Configurator or for manipulating the configuration session from a place other than the Siebel Configurator runtime UI.
  - The Remote Complex Object Instance Service is a business service that is available for accessing the Instance API.

NOTE: For more information on debugging the Configurator APIs, see see 2037936.1 (Doc ID) on My Oracle Support.

Instance APIs for the Complex Object Manager

Instance APIs for the Complex Object Manager follow these general rules:

- The Remote Complex Object Instance Service (RCOIS) is a business service. It can be accessed by anything in the Siebel architecture that can use a business service. As a business service, it is used by invoking methods, passing in property sets with input arguments, and getting results from the Outputs property set.

  NOTE: Before 7.8 the RCOIS was the API for using remote Siebel Configurator while Complex Object Instance Service (COIS) was the API for the embedded Siebel Configurator. In 7.8 and later versions the COIS and RCOIS are both proxies to the internal business service Siebel Configurator Service. All previous scripting efforts are still supported. Do all future scripting on the RCOIS.

- A session is uniquely identified by two ID values, the Object Id and the Root Id. In quotes, the Object Id is the Quote Id and the Root Id is the Quote Item Id for the top-level parent (the root). In assets, the Object Id and the Root Id are both the root Asset Id.
- A session is unique only within its own user session on a given Object Manager.
- A port is another name for a Relationship.
- A complex product is another name for a customizable product.
- The Port Id is the ID of the relationship as defined in the Complex Product Structure BusComp.
- The Prod Item Id is the ID of the relationship item as defined in the Complex Product Structure BusComp.
- The Path for an item is the Integration ID of the specific item.

- Version arguments are used only when testing a customizable product version that is different from the currently released version.

NOTE: The parameters are property set and, unless indicated, all properties are on the root level property set.

Instance APIs include the following methods:

LoadInstance Method

This method loads the complex object into memory. This is the starting point for all configurations.

You cannot use this method unless you have first called the LoadEAI method, which is part of the Copy business service. For more information, see the topic about the ISS Copy Service business service in *Siebel Order Management Infrastructure Guide*.

LoadInstance must always be called with Product Id as an additional parameter. This will certify that the application can find the correct remote Siebel Configurator object manager based on the Product Id.

NOTE: This method is part of the Complex Object Instance business service. It must not be confused with "LoadInstance Method" on page 424, which is part of Cfg Web UI Service Loader business service.

Input Arguments

- **ObjId.** The unique identifier of the complex object header (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row id).
- **IntObjName.** The name of the integration object specified in Oracle's Web Tools.
- **TriggerEvent.** The flag that determines if script events are triggered. Normally, it must be set to Y. Set it to N for special uses of the API where customizable product model script events are not desired.
- **(Optional) Product Id.** This parameter is used to find the correct remote Siebel Configurator object manager if Siebel Configurator object manager is enabled and used with component or server-based routing.

  NOTE: Product Id is mandatory if the Siebel Configurator is setup in remote mode, and also for customized workflows where the LoadInstance method is not the first method in the workflow.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **(Optional) NewRecord.** If set to Y, the instance will be populated with default values. The default is N.
- **(Optional) AutoSync.** If set to Y, the instance will be synchronized to the database immediately after loading. The default is N.
- **(Optional) SearchSpec.** Set this parameter to filter out all other hierarchical instances in the child buscomp. The default is an empty search specification. This parameter must have the following format:

  "[Header Buscomp.Id] = 'Id' AND [Item Buscomp.Root Id] = 'Root Id'"

  ex. [Quote.Id] = '10-4FR6D' AND [Quote Item.Root Id] = '10-81DUX'
- **(Optional) ExternalScript.** Set this parameter to Y when running headless configurations (for example, through Siebel COM Data Server). Anything other than the Cfg Web UI Service is considered headless configuration. The difference is based on the Siebel user who is calling the service. The default is N.
- **(Optional) ClearValues.** Y or N. Clears all cached Link Item Values. Use this parameter if you must clear values from the cache, for example because you the cached values are wrong and you must replace them.

Output Arguments

The following properties will be returned from the output property set:

- **CreateSession.** If this property is set to Y the method CreateSession must be called after LoadInstance. CreateSession will be used with the input parameter Links to pass a set of links to the method.
- **(Optional) IsConfig.** If this property is present and set to Y the configuration model has configuration rules defined.
- **(Optional) Links.** If this property is set to Y the model has linked items.

- **(Optional) Links.** If model has linked items, this parameter will hold linked items as child property set. This information shall be passed to CreateSession call in case there are any rules configured based on linked items.
- **(Optional) UnresolvedLinks.** If this property is set to Y the model has unresolved linked items that must be calculated by the caller.
- If NewRecord is set to Y in the input property set and CreateSession is set to N in the output property set, the output will have the instance property set returned as a child of type CxObj (see the output example below). Here is an example of a return property set:

```
< IsConfig='Y' UnresolvedLinks='Y' CreateSession='Y' Links='Y'>

<Links 1-19D0X='10/19/2001' 1-1Z876='SADMIN' 1Z771='SADMIN'>

</Links>"

      <UnresolvedLinks>"

       <UnresolvedLink DispName='Quote Name' Definition='<CfgVariableDef BUS_OBJ
= ""Quote"" BUS_COMP = ""Quote"" FIELD_NAME = ""Name"" SEARCH_SPEC = """"
SORT_SPEC = """" DEFAULT_VAL = """" EXECUTE = ""N""/>' Description='' DefValue=''
Name='Quote Name' BusObj='Quote' Field='Name' ID='1-1Z875' BusComp='Quote'>

         </UnresolvedLink>

      </UnresolvedLinks>

   </>
```

Extracting the instance property set from LoadInstance. The instance property set can be extracted by first getting the child property set of type CxObj and then extracting its only child.

Handling Links. A child property set of type Links is returned if the model has linked items. The Links child property set must then be extracted and passed in to CreateSession's OUTPUT arguments as a child property set. Since version 7.0.4 this changed to the INPUT property set for CreateSession. Configuration rules may have been defined for these linked items, so the configuration session must know the link values. The linked items are represented as property–value pairs with link IDs as properties and link values as property values, as in this example:

```
<Links 1-19D0X='10/19/2001' 1-1Z876='SADMIN' 1-1Z771='SADMIN'>
```

A child property set of type UnresolvedLinks is returned if the model has linked items that the business service could not resolve. The children of this property contain the information necessary to calculate the value of the linked item.

```
      <UnresolvedLinks>"

         <UnresolvedLink DispName='Quote Name' Definition='<CfgVariableDef BUS_OBJ =
""Quote"" BUS_COMP = ""Quote"" FIELD_NAME = ""Name"" SEARCH_SPEC = """" SORT_SPEC = """"
DEFAULT_VAL = """" EXECUTE = ""N""/>' Description='' DefValue='' Name='Quote Name'
BusObj='Quote' Field='Name' ID='1-1Z875' BusComp='Quote'>

         </UnresolvedLink>

      </UnresolvedLinks>
```

Only links that have the execute flag set or pull system parameters such as TODAY will be resolved by Siebel Configurator when used as headless configurations. The programmer must resolve all other links.

NOTE: Make sure the unresolved links are calculated and their IDs and values are added to the Links child property set as properties, with the link ID as the property and the link value as the property value.

CreateSession Method

This method initializes a configuration session, which is necessary for products with components that have constraint rules. It is called immediately following LoadInstance where required.

Input Arguments

- **ObjId.** The unique identifier of the complex object header (for example, Quote Id).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row id).
- **IntObjName.** The name of the integration object specified in Oracle's Web Tools.
- **TriggerEvent.** The flag that determines if script events are triggered. Normally, it must be set to Y. Set to N for special uses of the API where script events are not desired. LoadInstance and CreateSession must have the same setting.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **(Optional) NewRecord.** If set to Y, the instance will be populated with default values. The default is N.
- **(Optional) AutoSync.** If set to Y, the instance will be synchronized to the database immediately after loading. The default is N.
- **(Optional) ExternalScript.** This parameter must be set to Y when running headless configurations (for example, through Siebel COM Data Server). The default is N.

Output Arguments

If NewRecord is set to Y in the input property, the output will have the instance property set returned as a child of type CxObj. This is essentially the same output as the one that is returned from GetInstance**.**

SetInstance Method

This method creates a configuration session with the supplied property set, permitting configuration without directly writing to the database. The structure of the input property set does not need to correspond to a Siebel object, such as a quote that is indicated by the integration object specified.

When you use SetInstance, the input property set must be wrapped inside a parent property set.

For example, to use the following instance property set:

```
<PropertySet PrimaryRowId="42-56O78" OutputIntObjectName="7.7 Quote Integration
Object">

  ......

</PropertySet>
```

You must wrap it in another property set, as follows:

```
<PropertySet>

<PropertySet PrimaryRowId="42-56O78" OutputIntObjectName="7.7 Quote Integration
Object">

  ......

</PropertySet>

</PropertySet>
```

Input Arguments

Same arguments as LoadInstance but also requires the property set indicating the state to load. This property set must have the SiebelMessage object as the only first level child.

Output Arguments

Same arguments as LoadInstance.

SyncInstance Method

This method saves the complex object instance where it originated.

Input Arguments

- **ObjId.** The unique identifier of the complex object header.
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **IntObjName.** The name of the integration object specified in Oracle's Web Tools.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

None

UnloadInstance Method

This method removes the existing configuration session from memory. It must be called after synchronizing the instance at the end of the configuration session.

Input Arguments

- **ObjId.** The unique identifier of the complex object header.
- **RootId.** The unique identifier of the complex object root.
- **IntObjName.** The name of the integration object that was used to load the instance.

Output Arguments

None.

GetAllPorts Method

This method retrieves a list of all ports and (possibly) their contents for a product. It gets all ports for a product but not for its child products. It retrieves the basic definition of the product and does not consider any current configuration session state, so every possible port is retrieved.

Input Arguments

- **Product Id.** The ID of the product in Internal Product.
- **(Optional) Version.** Version is used only in validate mode.
- **GetPortDomain.** The flag that determines whether or not to also retrieve the domain of each port. Use Y or N to get the domain or not.

Output Arguments

All ports are returned as children of the output property set of type Port.

```
<GetAllPorts>

   <Port

      Class Id="<value>"

      MinCardinality="<value>"

      ClassName="<value>"

      MaxCardinality="<value>"

      Port Item Id="<value>"

      DefCardinality="<value>"

            Port Display Name="<value>"
```

```
      DefaultPortObjId="<value>"
      Name="<value>"
   ></Port>
   <Port
      Port Information here
   ></Port>
</GetAllPorts>
```

Port information can be returned through GetProperty(). For example, GetProperty(Class Id).

EnumObjects Method

This method returns either all immediate objects under an object or all immediate objects under a specified port. This gets the items that are currently in the port, not the items that could be there.

Input Arguments

- **ObjId.** The unique identifier of the complex object header (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row id).
- **IntObjName.** The name of the integration object specified in Oracle's Web Tools.
- **Parent Path.** The path to the parent object whose child objects you want to enumerate. The path is the object's Integration ID.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **(Optional) Port Item Id.** The ID of a specific port (for example, ORIG_ID of the port in S_PROD_ITEM table). If specified, only the items in this port are enumerated; otherwise, all items in all immediate ports are returned.

Output Arguments

Output can be in the form of two types of property sets, instance property sets and generic property sets.

Instance property sets return information about child items, as follows:

```
<Output>
   < Name=”value” Product Id=”value” Path=”value” Sequence Number=”value” />
   …
</Output>
```

Generic property sets send notifications to the user that there is a violation of the minimum or maximum required quantity for a one or more items within the relationship. A generic property set follows each instance property set, if there is a violation of the minimum or maximum required quantity for this instance.

GetAttribute Method

This method retrieves the value of an attribute.

Input Arguments

- **ObjId.** The unique identifier of the complex object root.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Path.** The path of the item where you are retrieving an attribute.
- **Name.** The attribute name.

Output Arguments

The value is returned as a property of the output property set, as follows:

```
<Output Value="value">

</Output>
```

GetFieldValues Method

This method retrieves field values for a product that exists in the complex product.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- Path – the path to the item.

Output Arguments

The output property set returned will have the field names as properties, as follows:

```
< Field="value" Field="value" … Field="value"/>
```

GetInstance Method

This method gets the loaded instance as a property set. It returns the full structure of products and attributes.

Input Arguments

- **ObjId.** The unique identifier of the complex object root.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

The entire property set output is the complex object instance.

GetParents Method

This method retrieves all the parents of an item.

Input Arguments

- **ObjId.** The unique identifier of the complex object root.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Path.** The path of the item.

Output Arguments

The property set returned will have child property sets, each with the following properties:

```
< >
   < Product Id=”value” Name=”value” Sequence Number=”value” Path=”value”/>
   …
</ >
```

GetPossibleDomain Method

This method retrieves selectable items from the configuration engine for a port

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Parent Path.** The item parent path.
- **Port Item Id.** The item port id.

Output Arguments

The property set returned will have the possible domain item product Ids as properties, each with the value 0, as follows:

```
< ProdId1=”0” ProdId2=”0” … ProdIdn=”0” />
```

GetPossibleValues Method

This method retrieves selectable values from the configuration engine for an attribute.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **IntObjName.** The name of the integration object.
- **Path.** The Integration ID of the port to which this attribute is attached.
- **XA Id.** The ID of the attribute for which the values need to be determined.

Output Arguments

The property set returned will have the possible values as the property names, as follows:

```
< [PossibleValue1]=”Val1” [PossibleValue2]=”Val2”/>
```

GetProductId Method

This method gets the root Product ID of the complex object instance.

Input Arguments

- **ObjId.** The unique identifier of the complex object header (for example, Quote ID).

- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

The product Id is returned as a property of the output property set, as follows:

```
< Product Id=″value″ />
```

GetRootPath Method

This method returns the path of the complex object instance root.

Input Arguments

- **ObjId.** The unique identifier of the complex object header (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

The root path is returned as a property of the output property set, as follows:

```
< Path=″value″ />
```

HasGenerics Method

This method returns generics and children flags for an item. A port has generics if the required cardinality is greater than the current cardinality and no default product is specified.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row id).
- **IntObjName.** The name of the integration object.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Port Item Id.** The ID for the port item that will have either children or generics.
- **Path.** The path to the parent item of interest.

Output Arguments

- **HasGenerics.** Y is present if it does, not present if it does not.
- **HasChildren.** Y is present if it does, not present if it does not.

GetConditionVal Method

This method allows users to call and retrieve the values of the named expression variables.

Input Arguments

- **CondName.** The name of the condition variable.
- **IntId.** The integration id of the product.

Output Arguments

If the condition variable is found and evaluates to true, the property Result is set to Y in the output arguments. If not, the Result is set to N.

For example, define a Product as follows:

Class1

-> Att1

Product: Child1 (associated with Class1)

Attribute: Att1 (which has the default value of one)

Constraints List:

Name: RuleOne

Constraint: The attribute SekAtt1 = one sets the value of the procedural condition variable to true

Product: Root (associated with Class1)

Re1: Child1

You can use the following script:

```
function Cfg_AttributeChanged (ChangedAttribute)
{
var Service = TheApplication().GetService("Configurator Service");
```

```
var Variable ="RuleOne";
var VariableVal = "";
var IntegrationID=  AddItem("$.[Root]#1", "R1", "Child1", "1")

var InputArgs = TheApplication().NewPropertySet();
var OutputArgs = TheApplication().NewPropertySet();

InputArgs.SetProperty("CondName", Variable);
InputArgs.SetProperty("IntId", IntegrationID);

Service.InvokeMethod("GetConditionVal", InputArgs, OutputArgs);
Variable =   OutputArgs.GetProperty("Result");

if (Variable == "Y")
{
    TheApplication().RaiseErrorText("True" );
}
else
{
    TheApplication().RaiseErrorText("False" );
}
}
```

Instance APIs to Interact with Conflicts and Messages

APIs to Interact with Conflicts and Messages include the following methods:

■ "UndoLastRequest Method" on page 446

NOTE: These methods apply only to products with components with constraint rules.

GetDetailedReqExpl Method

This method retrieves conflict messages.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **IntObjName.** The name of the integration object.

Output Arguments

Expl#. The explanations for the conflicts. Substitute a number for #, such as Expl0, Expl1, and so on.

GetExplanations Method

This method retrieves configuration explanations for an item.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Path.** The path of the item.

Output Arguments

The property set returned will have child property sets, each with the property Value as the explanation, as follows:

```
<Output>
   <Expl Value="Explanation"/>
   <Expl Value="Explanation"/>
   …
   <Expl Value="Explanation"/>
```

```
</Output>
```

GetSignals Method

This method retrieves configuration engine signals.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Path.** The integration ID where the item gets signals.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

The property set returned will have child property sets, as follows:

```
< >
   <Signal Expl=”signal”/>
   <Signal Expl=”signal”/>
   …
   <Signal Expl=”signal”/>
</ >
```

RemoveFailedRequests Method

This method removes all failed requests sent to the configuration engine.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

None.

UndoLastRequest Method

This method removes the last request sent to the configuration engine.

Input Arguments

- **ObjId.** The unique identifier of the complex object root (for example, Quote ID).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item row Id).
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

Output Arguments

None.

Instance APIs to Set Product and Attribute Values

The following methods are used for setting product and attribute values:

AddItem Method

This method adds an item to a specified port, creating a new instance of an item. If you want to change the quantity of an existing instance of an item, use SetItemQuantity.

Input Arguments

- **ObjId.** The unique identifier of the complex object header.
- **RootId.** The unique identifier of the complex object root.

- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **(Optional) AutoResolve.** Automatically resolves port cardinality violations. The default is N.
- **Prod Item Id.** Product Id. This is S_ISS_SUB_OBJ.SUB_OBJ_ID (in other words CFG_MODEL_ID of referred product).
- **Name.** The name of the item.
- **Product Id.** The product id in S_PROD_INT table.
- **Port Item Id.** Port or domain information. This is:
  - S_ISS_SUB_OBJ.PAR_REL_ID if this field is not null. Port has type Domain of class or dynamic class relationships
  - S_ISS_SUB_OBJ.ORIG_ID if S_ISS_SUB_OBJ.PAR_REL_ID is null. Port has type Product (S_ISS_SUB_OBJ.SUB_OBJ_TYPE_CD = Product)
- **Quantity.** The item quantity.
- **List Price.** The item list price from Pricing Manager, which can be empty.
- **Current Price.** The current price from Pricing Manager, which can be empty.
- **Parent Path.** The path is the object's Integration ID of the Parent Line Item where this additem call will be acted on.

Output Arguments

None.

CopyInstance Method

This method copies an instance.

Input Arguments

- **ObjId.** The unique identifier of the complex object header of the source instance.
- **RootId.** The unique identifier of the complex object root of the source instance.
- **DestObjId.** The unique identifier of the complex object header of the destination instance.
- **IntObjName.** The name of the integration object specified in Oracle's Web Tools.

Output Arguments

None.

GetLinkItemValues Method

This method retrieves the linked item values. Use it as follows:

1. Clear the cache values if specified by the caller.
2. For the search specification set by the caller:
   - If only RootProductId is given and no child property sets are given, try to find only the values for LinkItems that are under the given RootProductId.
   - If both RootProductId and Child Property Sets are given, try to find only the values for LinkItems that are defined in the child property sets.

Input Arguments

- **ClearValues.** Y or N. Clears all cached Link Item Values.
- **GetAllValues.** Y or N. Gets all values from the cache or only the values passed in for the search specification.
- **RootProductId.** Root Product Id.
- **LIVersionId.** Root Product Version Id (optional).
- Child Property Set (0) Type: LinkedItem:
  - ProductId. Product Id
  - VersionId. (optional) Version Id
  - InClass. Y or N.
  - LinkItemId. Link Item Id
- Child Property Set (1) Type: LinkedItem:
  - ProductId
  - VersionId (optional) Version Id
  - InClass. Y or N.
  - LinkItemId. Link Item Id

Output Arguments

Each linked item is returned as a child property set as follows:

- Child Property Set Type: LinkedItem:
  - CxLnkItmId. Link Item Id.
  - CxLnkItmName. Link Item Name.
  - CxLnkItmDesc. Description.
  - CxLnkItmDef. Definition.
  - CxLnkItmIsActive. IsActive Flag.

- CxLnkItmValue. Value.
- CxLnkItmDisplayValue. Display Value.
- CxLnkItmType. Type.
- CxLnkItmProductId. Product Id where this linked item is defined.
- CxLnkItmVersionId. Version Id of the previous product.

RemoveItem Method

This method removes an item from the instance.

Input Arguments

- **ObjId.** The unique identifier of the complex object header.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used only when testing a customizable product version that is different from the currently released version.
- **Path.** The path of the item.

Output Arguments

None.

ReplaceItem Method

This method replaces an existing item with the new item on a specified port, removing the existing item from the port and creating a new instance for new item.

Input Arguments

- **ObjId.** The unique identifier of the complex object header.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used only when testing a customizable product version that is different from the currently released version.
- **Path.** Integration Id of the existing product (which is going to be replaced by new product)
- **Name.** The name of the new item.
- **Product Id.** The new product id in S_PROD_INT table.
- **Prod Item Id.** The ID of the new item (for example, ORIG_ID in S_PROD_ITEM table).
- **Port Item Id.** The ID of the item's port (for example, ORIG_ID of the port in S_PROD_ITEM table).

- **Quantity.** The new item quantity.
- **(Optional) AutoResolve.** Automatically resolves port cardinality violations. The default is N.
- **(Optional) List Price.** The item list price from Pricing Manager, which can be empty.
- **(Optional) Current Price.** The current price from Pricing Manager, which can be empty.
- **(Optional) Parent Path.** Integration Id of the parent item the port belongs to.
- **(Optional) Parent Display Name.** Display name of the parent product. This parameter and the display parameters that follow would be used in logging error messages in case of error.
- **(Optional) New Child Display Name.** Display name of the new child product.
- **(Optional) Port Display Name.** Display name of the Relationship of the child product.

Output Arguments

None.

RepriceInstance Method

This method updates the instance with values from the Pricing Manager service. A call to the Pricing Manager service's CalculatePriceCX method returns a property set that is the input to this method.

Input Arguments

- **ObjId.** The unique identifier of the complex object header.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.

The input property set has child property sets containing the repricing information. The format of the input property set is as follows:

```
< ObjId="value" RootId="value" Version="value" >
   < IntId="integration id" FieldName="value"… FieldName="value" >
   < IntId="integration id" FieldName="value"… FieldName="value" >
    …
< />
```

In this context, the Integration ID is used to retrieve the instance item.

Output Arguments

None.

SetAttribute Method

This method sets the value of an item's attribute.

Input Arguments

- **ObjId.** The unique identifier of the complex object root.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Path.** The path of the item with attribute to set.
- **Value.** The attribute value.
- **Name.** The attribute name.
- **XA Id.** The extended attribute ID. This is the row ID of the attribute in the XA Attribute business component. This parameter is required, and you can use a dummy value instead of the actual row ID.
- **Property Type Code.** The attribute type.

Output Arguments

None.

SetItemQuantity Method

This method sets the quantity of an item.

Input Arguments

- **ObjId.** The unique identifier of the complex object root.
- **RootId.** The unique identifier of the complex object root.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version.
- **Path.** The path of the item.
- **Quantity.** The quantity to set.

Output Arguments

None.

SetFieldValue Method

This method updates the value of a field in a line item.

Input Arguments

- **Path.** The Integration Id of the line item (or of the parent line item in the case of an attribute).
- **FieldName.** The field name. This must not be a field that is used and controlled by Siebel Configurator (such as: Quantity, Port Item Id, Integration Id, Attribute Value, and so on).
- **XA Id.** (Optional) The attribute Id. Specify only if you are setting the value in an attribute field.
- **Value.** The value of the field.
- **ObjId.** A unique identifier of the complex object header (for example, quote ID).
- **RootId.** A unique identifier of the complex object root (for example, quote line item row ID).

Output Arguments

None.

SetLinkItemValues Method

This method overwrites the linked item values for the specified linked items. It can be called multiple times and the effects are cumulative; when updating the same linked item, the latest call will supersede the previous calls.

NOTE: For more information, see Article ID 2251129.1 on My Oracle Support.

Input Arguments

This can contain many child Property Sets of Type SetLinkItemValues. The Value of the child Property Set is the Product Id whose linked items need to be overwritten. The properties of each child Property Set specify the linked item name and value as the name value pairs.

For example:

```
InputPropSet

Child PropSet (0) Type=LinkItemOverwrite Value=Prod1

   Account City = San Francisco

   Account Country = USA

Child PropSet (1) Type=LinkItemOverwrite Value=Prod2

   Account = ABC Inc.

//                  ---------  Child Property Set

//                           Type : SetLinkItemValues
```

```
//                          Value :Product Id 1 value
//                          Name - Value Pairs (Name is the Link Item Name.
//                                              Value is the value you want to set.)
//                  ---------  Child Property Set
//                          Type : LinkItemOverwrite
//                          Value :Product Id 2 value
//                          Name - Value Pairs
//                  ---------  Child Property Set
//                          Type : LinkItemOverwrite
//                          Value :Product Id 3 value
//                          Name - Value Pairs
```

Object Broker Methods

The following methods call the Cfg Object Broker business service, which functions as a wrapper for the Object Broker:

ResetSKC Method

This method posts a request to invalidate all cached object definitions. The application does this logically by going through the cached records in the ISS VOD Cache Sync business component and resetting the fields values for all the VOD objects versions that are recorded as been cached. For more information about how this method is invoked, see "Clearing the Siebel Configurator Cache to Improve Performance" on page 45.

Input Arguments

Not applicable.

Output Arguments

Not applicable.

ResetObjInSKC Method

This method posts a request to invalidate a specific cached object definition. The application does this logically by querying the ISS VOD Cache Sync business component to find the record based on the Type and VodId input parameters. If a record is found then the field values for that specific record are reset.

Input Arguments

- **Type.** The type of the VOD object. Valid values are: ISS_ATTR_DEF, ISS_CLASS_DEF or ISS_PROD_DEF.
- **VodId.** The value of the VOD object Id. This value must match the value found in either the S_VOD.OBJECT_NUM or the S_PROD_INT.CFG_MODEL_ID field.

NOTE: Both input arguments are required.

Output Arguments

Not applicable.

GetProdStruct Method

This method returns the full structure of the customizable product.

Input Arguments

- **RootId.** The unique identifier of the complex object root. If provided, RootName, Vendor, and Org are ignored. If not provided, RootName, Vendor, and Org are used to uniquely identify the product.
- **RootName.** The root product name. Optionally, you can use Name together with Vendor and Org to uniquely identify a product.
- **(Optional) Version.** Version arguments are used *only* when testing a customizable product version that is different from the currently released version. Specify 0 to return the work space.
- **(Optional) Vendor.** Use with RootName to uniquely identify the product. By default, this is empty.
- **(Optional) Org.** Use with RootName to uniquely identify the product. By default, this is empty.
- **Full.** Y returns the full product structure. N returns the first level of the product.

Output Arguments

The output is in Property Set format.

```
<ProdStruct> RootId

   <ProdId> Name ClassId
```

```
      <Port> Name ClassName ClassId OrigId Type MinCard MaxCard DefltCard LocalType
InternalType
         <Subobject Id/>
         ...
      </port>
      ...
      <Attribute> Name
         <Domain Value />
         ...
      </Attribute>
      ...
   </ProdId>
   ...
</ProdStruct>
```

DeltaQuote Method

This method performs a recursive tree comparison of two property sets to determine the difference between them based on supplied criteria. It returns a copy of the destination product instance, marked up to indicate changes. This preconfigured API is called from an external service only in SIS Order Management, which is the only way to see this API function for the API Discovery.

In one example, you start with a computer that has one hard drive and a 900 MHz processor. You upgrade it to add a second hard drive (quantity now = 2) and replace the processor with a 1000 MHz model. The result would be one existing hard drive, one new hard drive (the instance is split), one 900 MHz CPU removed, and one new 1000 MHz CPU.

If the output property set is empty and no error code is thrown, the most likely cause is that the instances were not recognized. Check the RootId parameters and the indenting of your source and destination SiebelMessages.

Input Arguments

- **SrcRootId.** The root ID for the product in the source of the comparison. This is the “before” property set.
- **DestRootId.** The root ID for the product in the destination of the comparison. This is the “after” property set.
- **DeltaSrcField.** The Path field in the source instance.
- **DeltaDestField.** The Path field in the destination instance.

- **SrcItemIntComp.** The name of the integration component for the items in the source instance (for example, Quote Item).
- **DestItemIntComp.** The name of the integration component for the items in the destination instance (for example, Quote Item).
- **SrcXAIntComp.** The name of the integration component for the XA in the source instance (for example, Quote Item XA).
- **DestXAIntComp.** The name of the integration component for the XA in the destination instance (for example, Quote Item XA.)
- **ITEM_MAPPING.** This is a property set with type= "ITEM_MAPPING." It contains a list of those fields that must be copied when creating a new instance of an item in the destination property set. The property names are the fields in the source instance, and the property values are the names in the destination instance. Here is an example output from the API sniffers:

```
CHILD PROPERTY SET 3
      Type: ITEM_MAPPING Value:
      Unit Price = Unit Price
      Action Code = Action Code
      Root Id = Root Id
      Port Item Id = Port Item Id
      Integration Id = Integration Id
      Discount Amount = Discount Amount
      Parent Id = Parent Id
      Product Id = Product Id
      Prod Item Id = Prod Item Id
      Quantity = Quantity
```

- **XA_MAPPING.** This is a property set with type= "XA_MAPPING." It contains a list of those fields that must be copied for each of the attributes when creating a new instance of an item in the destination property set. The property names are the fields in the source instance, and the property values are the names in the destination instance. Here is an example output from the API sniffers:

```
CHILD PROPERTY SET 2
      Type: XA_MAPPING Value:
      Action Code = Action Code
      Value = Value
      Read Only = Read Only
      Name = Name
      Property Type Code = Property Type Code
      XA Id = XA Id
```

- **ITEM_COMPARE.** This is a property set with type= "ITEM_COMPARE." It defines what constitutes a unique instance of an item in the source and destination instances. For the service, it answers the question "How do I know if these two things are the same?" Here is an example output from the API sniffers:

```
CHILD PROPERTY SET 1
      Type: ITEM_COMPARE Value:
      Port Item Id = Port Item Id
      Product Id = Product Id
```

- **XA_COMPARE.** This is a property set with type= “XA_COMPARE.” It defines what constitutes a unique instance of an attribute in the source and destination instances. For the service, it answers the question “How do I know if these two things are the same?” Here is an example output from the API sniffers:

```
CHILD PROPERTY SET 0
      Type: XA_COMPARE Value:
      Value = Value
      Name = Name
      Property Type Code = Property Type Code
      XA Id = XA Id
```

- **SrcInst.** This is the Source Instance (“before”) that will be used in the delta comparison. It is a double-indented SiebelMessage with a type of SrcInst. The first section of the output from the API sniffers is shown here for reference. Note the indenting of the SiebelMessage.

```
CHILD PROPERTY SET 4
      Type: SrcInst Value:
      CHILD PROPERTY SET 0
         Type: Value:
         CHILD PROPERTY SET 0
            Type: SiebelMessage Value:
            MessageId = 1-12949
            IntObjectFormat = Siebel Hierarchical
            MessageType = Integration Object
            IntObjectName = CX Product Validation
            CHILD PROPERTY SET 0
               Type: ListOfCX Product Validation Value:
               CHILD PROPERTY SET 0
                  Type: Product Header Value:
                  Name = 1-12950
                  Price List Id = 1-ZEC
                  Id = 1-12951
                  CHILD PROPERTY SET 0
                    Type: ListOfProduct Item Value:
                    CHILD PROPERTY SET 0
                       Type: Product Item Value:
                       Action Code = Existing
                       Port Item Id =
                       Integration Id = 1-12744
                       Cfg Type = Configurator
                       Name = System Chassis
                       Product Id = 1-1M9Y
                       Prod Item Id = null
                       Quantity = 1.0
                       Id = 1-12744
```

- **DestInst.** This is the Destination Instance (“after”) that will be used in the delta comparison. It is a double-indented SiebelMessage with a type of DestInst. The first section of the output from the API sniffers is shown here for reference. Note the indenting of the SiebelMessage.

```
CHILD PROPERTY SET 5
      Type: DestInst Value:
      CHILD PROPERTY SET 0
         Type:  Value:
```

```
CHILD PROPERTY SET 0
   Type: SiebelMessage Value:
   MessageId = 123
   IntObjectFormat = Siebel Hierarchical
   MessageType = Integration Object
   IntObjectName = CX Product Validation
   CHILD PROPERTY SET 0
      Type: ListOfCX Product Validation Value:
      CHILD PROPERTY SET 0
         Type: Product Header Value:
         Name = 1-12950
         Price List Id = 1-ZEC
         Id = 1-12951
         CHILD PROPERTY SET 0
            Type: ListOfProduct Item Value:
            CHILD PROPERTY SET 0
               Type: Product Item Value:
               Has Generics Flag = Y
               Action Code = Existing
               Integration Id = 1-12744
               Port Item Id =
               Sequence Number =
               Name = System Chassis
               Cfg Type = Configurator
               Product Id = 1-1M9Y
               Quantity = 1.0
               Prod Item Id = null
               Id = 1-12744
```

Output Arguments

The destination instance is returned as a SiebelMessage, modified and marked up with status information. The status (new, modified, existing, removed) is indicated in the Action Code field of each item and attribute.

Instance APIs to Select the Siebel Configurator User Interface

The following methods allow you to select the Siebel Configurator User Interface:

- “SelectCfgUIService Method” on page 459
- “SetUIOption Method” on page 460.

These methods are part of the Cfg Web UI Service Loader business service.

SelectCfgUIService Method

Syntax

SelectCfgUIService (inputArgs as Property Set, outputArgs as Property Set)

Input Arguments

- **CxVersion.** The version of the product that you are validating or customizing. Empty means latest version.
- **ComplexProductId.** The product id (in S_PROD_INT) for the product that you are validating or customizing.

Output Arguments

- **Cfg UI Option Id.** Optional. If empty, Siebel Configurator will use the default templates.
- **High Interactivity.** Required. The value is Y.

Usage

For any method you call on the UI service, you can call SelectCfgUIService to determine which UI Service to load.

You must specify the name of your business service in the User Property of the UI Service (Cfg Web UI Service (JS) and Cfg Web UI Service) as follows:

```
Loader Service Name = <your business service name>
```

You must also modify all the related workflows (such as Validate and Customize) to first call this customized method in your business service.

Example

For example, you can use the LoadInstance method to call SelectCfgUIService as follows:

```
function LoadInstance (inputArgs, outputArgs)

{

   var oUIService;

   var oService =TheApplication().GetService("Cfg Web UI ServiceLoader");

   var oOutputs =TheApplication().NewPropertySet();


   oService.InvokeMethod("SelectCfgUIService", inputArgs, oOutputs);


   var optionId      = oOutputs.GetProperty ("Cfg UI Option Id");
```

```
    var strSupportJS  = oOutputs.GetProperty ("High Interactivity");


    with(inputArgs)
    {
        SetProperty ("Cfg UI Option Id",    optionId);
        SetProperty ("High Interactivity", strSupportJS);
    }
    [if (strSupportJS == "Yes")]
    {
        oUIService = TheApplication().GetService ("Cfg Web UI Service (JS)");
    }
    else
    {
        oUIService = TheApplication().GetService ("Cfg Web UI Service");
    }
    oUIService.InvokeMethod("LoadInstance", inputArgs, outputArgs);
}
```

SetUIOption Method

SetUIOption method is used to display Siebel Configurator interface.

Syntax

SetUIOption (inputArgs as Property Set, outputArgs as Property Set).

Input Arguments

A property set that contains the name or value pairs representing the value of the Product Id and UI Option Name fields.

Usage

Because this method is part of a cached Business Service, you can call this method at any time before Configurator is called.

The values you use as input arguments indicate which UI option to use for each product you specify.

Example

Add a workflow step at the beginning of the Siebel Configurator Load Workflow. This step calls a custom business service that determines which UI option to use for the Product that is being customized. After determining which UI option to use, the workflow uses the SetUIOption method to set the appropriate UI option.

The sample custom business service has the following features:

- Business Service Name: SIS OM Workflow Utility
- Business Service Method Name: Force_UI

Business Service Method Arguments

Argument Name	Data Type	Storage Type	Type
Product_Id	String	Property	Input
UI_Name	String	Property	Input

Business Service Server Script:

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)

{


   Outputs.SetProperty("ErrorCode", "");

   Outputs.SetProperty("ErrorText", "");

   Outputs.SetProperty("Error Code", "");

   Outputs.SetProperty("Error Message", "");


   switch(MethodName)

   {

      case "Force_UI":

         var strProdId = Inputs.GetProperty("Product_Id");

         var strUI_Name = Inputs.GetProperty("UI_Name");

         var oService = TheApplication().GetService("Cfg Web UI Service Loader");

         var oInputs = TheApplication().NewPropertySet();

         var oOutputs = TheApplication().NewPropertySet();


         with(oInputs)
```

```
        {
        /* The input argument name is passed as a string (the product id)*/
   /*************************************************************/
        SetProperty(strProdId,strUI_Name);
        }
        /* Force Configuration session UI */
        /*******************************/
        oService.InvokeMethod("SetUIOption", oInputs, oOutputs);
        return (CancelOperation);
        break;

      default:
        return (ContinueOperation);
        break;
   }
}
```

Scenario for the Example

In the previous code, consider the scenario where product KG1 and KG2 are to be displayed. Assume both products have the following properties.

Argument Name	Data Type	Storage Type
UI Options	Open UI Option 1, Open UI Option 2, Open UI Option3	Open UI Option 1, Open UI Option 2, Open UI Option3
Product Id	12-12345	12-67890

To set Open UI Option 1 for KG1 and Open UI Option 3 for KG2, replace the code SetProperty(strProdId,strUI_Name) with the following lines:

```
SetProperty("12-12345", "Open UI Option 1");
SetProperty("12-67890", "Open UI Option 3");
```

Usage for the Example

In the workflow that launches the Siebel Configurator UI (which is Configurator Load for non-ABO or SIS OM Edit for ABO), the customer might add two steps before launching Siebel Configurator:

- One to invoke the sample script
- One to invoke the SetUIOption method to preset the UI option

Then Siebel Configurator uses this preset option during end-user configuration sessions.

Instance API to Validate Customizable Products

The following method allows you to batch validate customizable products.

BatchValidate Method

There are three options for using this method:

- LoadInstance option where the instance is already loaded in the context service. This is for validating an existing quote, order, agreement, or asset. A workflow step to load the instance into the context service must be done before calling BatchValidate.
- LoadInstance option where the BatchValidate API will handle loading the instance into the context service. This is for validating an existing quote, order, agreement, or asset. The instance does not have to be in the context service already.
- SetInstance option. This option allows the user to pass in the customizable product instance as a property set.

Input Arguments

The following are the input arguments:

- **Mode.** Quote, Order, Agreement, or Asset.
- **IntObjName.** The name of the integration object specified in Oracle's Web Tools (for example, 7.7 Quote Integration Object, 7.7 Order Entry Integration Object, 7.7 Asset Integration Object, 7.7 Service Agreement Integration Object, SIS OM Asset, SIS OM Quote, SIS OM Order).
- **ProductId.** The unique identifier of the product. This ID is required when using Siebel Configurator with remote enabled.
- **ObjId.** The unique identifier of the complex object header (for example, Quote Id, Asset Id, Order Id).
- **RootId.** The unique identifier of the complex object root (for example, Quote Line Item Id, when Mode = Asset, Obj Id and Root Id point to Asset Id).
- **InstanceName.** The name of the instance being pushed to the context service. The value of this argument must be:
  - The name of the instance in the context service if using Option 1
  - Empty if using Option 2
  - Optional for Option 3

- **SetInstance.** (Optional) Specifies whether to use the SetInstance or LoadInstance option. Value can be Y or N. If set to Y, the first child property set of the input argument must be the CP instance being validated in its property set representation form. N by default.
- **DetailedExpl.** (Optional) Specifies whether to return a detailed explanation if there's a conflict. Value can be Y or N. N by default.
- **AutoSync.** (Optional) Specifies whether to synchronize changes back to the database. Value can be Y or N. N is the default. The output property set does not synchronize to the database if the BatchValidate Status is Invalid.

  NOTE: The AutoSync option must not be used for asset-based ordering. Instead, use delta processing, like the asset-based ordering workflow. For more information, see the workflow reference in *Siebel Order Management Guide*.
- **DisablePricing.** (Optional) Specifies whether to do pricing or not. Value can be Y or N. N by default.
- **OutOriginalInstance.** (Optional) Specifies whether to return the original instance as a property set. Value can be Y or N. Y by default.
- **OutCompletedInstance.** (Optional) Specifies whether to return the validated instance as a property set. Value can be Y or N. Y by default.
- **OutDelta.** (Optional) Specifies whether to return a delta quote representing the difference between the original and validated instance as a property set. Value can be Y or N. Y by default.
- **IgnoreNewOptionalAttr** (Optional). Specifies whether to ignore Optional Attributes. If the value is Y, then the output status is Valid even if optional attributes are omitted. If the value is not Y, then the status is Incomplete and an error message is displayed if optional attributes are omitted.
- **ExternalScript** (Optional). Set this parameter to Y when Linked Items must be calculated. The default is N

Output

The output of BatchValidate is a Siebel Message containing:

- **Batch Validation Status.** This property returns the status of BatchValidate. The return values can be:
  - Valid. No violations were found.
  - Incomplete. The constraint engine has found and corrected errors, but does not display an error message. The engine returns Invalid only if it can make a clear decision about how to correct the errors, which is the case for:
    - All violations of the maximum quantity where the engine detects unnecessary child products and deleting these child products brings the quantity below the maximum.
    - All violations of the minimum quantity where there is only one child product (that is, Domain = Product or Domain = Class/Dynamic Class where the domain has one product as a member).

- Invalid. The engine does not have enough information to correct errors. This is true of violations of the minimum quantity where the there is more than one child product that can be used to correct the error.

- **Original Instance.** A property set representation of the original instance being validated.
- **Completed Instance.** A property set representation of the validated instance.
- **DeltaQuote.** A property set representation of the difference between the original instance and the completed instance.
- **Generic Info.** A property set containing generics and children relationship information for all ports. For example, batch validate the following three children as part of a parent product:

  ```
  Parent prod

      Child prod 1

      Child prod 2

      Child prod 3
  ```

  The generic information in batch validate output is as follows:

  ```
  <Generic_spcInfo>

  <<_6SIA-4HSO8 _6SIA-4HQ0M="01" _6SIA-4HQ0L="01" _6SIA-4HQ0K="01" />

  <<_6SIA-4HSR5 />

  <<_6SIA-4HSR6 />

  <<_6SIA-4HSR7 />
  ```

  where:

  - 6SIA-4HSO8 is the order line item ROW_ID of the Parent Product.
  - 6SIA-4HQ0M, 6SIA-4HQ0L, 6SIA-4HQ0K are the relationship ROW_ID's for the child products.

  The code mapped against the relationship ROW_ID's indicates whether the relationship has generics and children.

  - When 1st bit is set, relationship has generics
  - When 2nd bit is set, relationship has children

  for example:

  - **01.** no generics, has children
  - **11.** has generics, has children

  Use this code to determine minimum cardinality violations.

- **MissingRequiredAttr.** A property set returning required attribute violations.

- **Error Message.** A property set containing the error messages, if any, encountered during batch validation.

NOTE: Eligibility & Compatibility is automatically enforced by BatchValidate based on the EligibilityDisplayMode parameter setting on Object Manager. There is no additional parameter needed to enforce eligibility and compatibility during batch validate.

In addition, if PRESERVE_ENGINE_AND_USER_PICKS environment variable is enabled, the value in the CFG_STATE_CD column will effect the Batch Validate result. The PRESERVE_ENGINE_AND_USER_PICKS environment variable is enabled by default from 8.1 onwards.

For information about the user pick and engine pick setting in CFG_STATE_CD column, see 477087.1 (Doc ID) on My Oracle Support. This document was previously published as Siebel Alert 749.

Reset CFG_STATE_CD for all line items and attributes to user pick or leave empty if you want to preserve the content in the input instance.

Sample Code

The following code is an example of how BatchValidate can be used.

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)

{


if (MethodName == "BatchValidate_Quote")

{

   var head = "8SIA-81CQ4"; //QuoteId

   var item = "8SIA-81CVG";  // Line Item Id

   var io = "7.7 Quote Integration Object"; // Integration Object


   var s = TheApplication().GetService("Remote Complex Object Instance Service");

   var inp = TheApplication().NewPropertySet();

   var out = TheApplication().NewPropertySet();


   inp.SetProperty("Mode","Quote");

   inp.SetProperty("IntObjName", io);

   inp.SetProperty("ObjId",head);

   inp.SetProperty("RootId",item);

   inp.SetProperty("DetailedExpl","Y");
```

```
    s.InvokeMethod("BatchValidate",inp,out);

    var s2 = TheApplication().GetService("EAI XML Write to File");
    var inp2 = TheApplication().NewPropertySet();
    var out2 = TheApplication().NewPropertySet();

    inp2.SetProperty("FileName","c:\\ab.xml");
    inp2.AddChild(out);
    s2. InvokeMethod("WritePropSet", inp2, out2);

    TheApplication().RaiseErrorText("Finished ...");
}
```

27 Siebel Configurator Version 6.x, 7.0 and 7.5

This chapter provides a conceptual mapping of Siebel Configurator version 6.x to version 7.0 and 7.5. Users of Siebel Configurator version 6.x must use this chapter to help them understand the similarities and differences between 6.x and version 7.0 and 7.5 Siebel Configurator.

The mappings are intended to present features that are functionally similar. That one feature maps into another does not mean that the version 7.0 and 7.5 feature is exactly equivalent or works in exactly the same way.

This chapter includes the following topics:

NOTE: This chapter does not compare version 6.x with version 7.8, because there is no direct upgrade path from version 6.x to version 7.8. The upgrade from version 7.0 or 7.5 to version 7.8 is a direct upgrade, which is not visible to the user.

Upgrading Version 6.x Models to Version 7.0 and 7.5

Upgrading version 6.x models to version 7.0 and 7.5 requires planning and a thorough understanding of the upgrade process. For information about data migration considerations, see 477067.1 (Doc ID) on My Oracle Support. This document was previously published as Siebel Technical Note 428.

Managing Models in Version 6.x and 7.x

The concept of a model in version 6.x maps in version 7.0 and 7.5 to a customizable product with a work space. Table 50 maps version 6.x features to version 7.0 and 7.5 for managing models.

Table 50. Managing Models

Version 6.x	Version 7.0 and 7.5	Comment
Create a new model	Create a customizable product with a work space.	None.
Delete a model	Delete customizable product structure.	Reverts customizable product to a simple non-configurable product.
Copy a model	Copy a customizable product.	None.
Share a model	Add a customizable product to another customizable product.	None.
Lock a model	Lock a customizable product work space.	None.
Import a model	Import a customizable product.	None.
Export a model	Export a customizable product.	Only the latest released version can be exported.
Export a model version	Only the latest released version can be exported.	None.
Validate a model	Validate a customizable product.	None.
Release a model	Release a customizable product.	None.
Revert to released model	Revert to a released customizable product.	None.
Model versions	Customizable product versions.	None.
Model synchronization	Customizable product synchronization.	None.
Associating a model with a product	Not applicable	The structure of a customizable product is stored with the product definition. Associating a model with a model-product is no longer required.
Required start date	Customizable product required start date	The customizable product version does not become available to users until the specified date.

Designing the Catalog in Version 6.x and 7.x

The tree structure of catalogs and items in version 6.x maps in version 7.0 and 7.5 to a hierarchy of relationships within the customizable product. A relationship is roughly equivalent to a category and functions as a named group that contains one or more items. Table 51 maps version 6.x features to version 7.0 and 7.5 for designing a catalog.

Table 51. Designing the Catalog

Version 6.x	Version 7.0 and 7.5	Comment
Design a tree structure	Create a hierarchy of relationships within a customizable product.	The hierarchy defines component relationships rather than being a grouping mechanism. Relationship definition includes cardinality (maximum quantity, minimum quantity, default quantity).
Create a category	Create a relationship.	A relationship is a named part of a customizable product. Relationships contain one or more items.
Add a product to a catalog	Add a product to a relationship.	None.
Set item sequence in a catalog	Set sequence of items in Structure view and sequence of group in Product UI Designer.	None.
Hide items in the catalog	Can hide an item by not adding it to a UI group in the Product UI Designer.	All products included in a customizable product, plus all attributes, resources, and links can be made visible to users.
Show all excluded items	Select UI control in Product UI designer that displays all items.	Excluded items are unavailable.
Create virtual product	Replaced with hidden attributes.	Virtual product functionality can be created by defining product attributes and then marking them hidden. Hidden attributes do not appear in quotes, orders, or agreements.

Working with Properties in Version 6.x and 7.x

The properties feature has been replaced in version 7.0 and 7.5 with attributes. Attributes are product characteristics that are defined for product classes. All products belonging to a class inherit the attributes of the class. Subclasses inherit the attributes of the parent class. You put products into the class hierarchy by assigning a class name in the product record. Attributes cannot be defined directly on a product. They must be inherited from the class to which the product belongs.

When you define an attribute, you can define the allowed values for the attribute. You can specify the allowed values, called an attribute domain, using a list of values or a range of values. The administrator can then set this value for an individual product so that it cannot be changed by the user.

Defining an attribute and setting its value so that it cannot be changed is functionally equivalent to defining a property and assigning it a value in version 6.x.

Working with Resources in Version 6.x and 7.x

Resources are implemented in the same fashion for customizable products in version 7.0 and 7.5 as they were for models in version 6.x. You define resources in the Resource Designer and then write provide rules and consume rules that adjust the value of the resource. In version 7.0 and 7.5, you can also provide or consume amounts from a product attribute.

In version 7.0 and 7.5, to enforce a resource or attribute so that its value does not drop below zero, you write a configuration rule that constrains the range of allowed values.

Working with Linked Items in Version 6.x and 7.x

In version 7.0 and 7.5, you define links within the context of a specific customizable product. You do not define links in a single location and then associate the definition with a model, as in version 6.x. Link definitions are added to a picklist so that you can add the definition to other customizable products. If you delete a link definition from the only customizable product in which it occurs, it is deleted from the picklist.

The things for which you can define a link, have not changed in version 7.0 and 7.5. You can define links on Siebel business components, the system date/time, and on the login ID of the current users.

Designing Rules and Logical Expressions in Version 6.x and 7.x

The version 6.x Basic Rule Designer, Logic Designer, and Advanced Rules Designer have been replaced by the Rule Designer in version 7.0 and 7.5. The Rule Designer provides a series of natural-language rule templates that you can use to create rules of any complexity. You can also create and save your own rule templates.

The logical operators in the version 6.x Logic Designer are provided in picklists associated with the rule templates.

A special rule template is provided to enter rules in Siebel Configurator Rule Language (renamed Rule Assembly Language in version 7.0 and 7.5). The operators and syntax in version 6.x Siebel Configurator Rule Language are supported in version 7.0 and 7.5. Table 52 maps version 6.x features to 7.0 and 7.5 for designing rules.

Table 52. Designing Rules in the Rule Designer

Version 6.x	Version 7.0 and 7.5	Comment
Create rule in Basic Rule Designer	Create rule using rule template in Rule Designer.	Rules can be created with or without conditions.
Create expressions in Logic Designer	Use logical operators associated with rule templates in Rule Designer.	Rules can be created with or without conditions.
Create rule in Advanced Rules Designer	Enter rule into special template in Rule Designer.	version 6.x operators and syntax are supported in version 7.0 and 7.5.
Category-to-product rules	Class-to-product rules.	The product class must be part of a relationship.
Category-to-category rules	Class-to-class rules.	The product classes must part of relationships.
Product-to-product rules	Product-to-product rules.	None.
Copy and delete rules	Copy and delete rules.	Includes rules that are logical expressions.
Activate and deactivate rules	Activate and deactivate rules.	Includes rules that are logical expressions.
Application generates rule explanations	Application generates rule explanations or you can write explanations.	Includes rules that are logical expressions.
Rule Summary report	Rule Summary report.	None.

Table 52. Designing Rules in the Rule Designer

Version 6.x	Version 7.0 and 7.5	Comment
Enforce resource total by placing check mark in resource record	Write rules to prevent resources from having negative values.	None.
Syntax checker for Siebel Configurator Rule Language	Syntax checking is provided when building rules using the Assisted Advanced Rule template and all other rule templates.	Siebel Configurator Rule Language is called Rule Assembly Language in version 7.0 and 7.5.

Designing Scripts in Version 6.x and 7.x

In version 7.0 and 7.5, the full Siebel API is accessible from within a script. See Siebel API documentation for more information on the Siebel API. Because the Siebel API is available, the number of Siebel Configurator-specific events and functions has been reduced in version 7.0 and 7.5.

In addition, the method for associating scripts with parts of a model has changed. In version 7.0 and 7.5, you associate a script with an item by writing the script on an event called for the product root. The event returns a matrix of records, one for each item that has changed in the solution. An item can be any product added to the customizable product from the product table. Events do not return changes to relationship quantities or resources.

If a customizable product contains other products with components, another event is provided so you can write scripts on the child customizable product directly.

The Script Designer in version 7.0 and 7.5. does not provide a hierarchical tree display of the customizable product. In version 7.0 and 7.5, the Cfg ID of an item can no longer be passed as an argument. Instead, the name of the item as a string is passed. A name syntax is provided to allow you to uniquely specify a product name. Table 53 maps version 6.x features to version 7.0 and 7.5 for designing scripts.

Table 53. Designing Scripts

Version 6.x	Version 7.0 and 7.5	Comment
Create scripts	Create scripts in the Script Designer.	None.
Script inheritance	Scripts are not associated directly with relationships and are not inherited.	None.
Copy, edit, and delete scripts	Copy, edit, and delete scripts.	None.
Siebel Visual Basic and Siebel eScript languages	Siebel Visual Basic and Siebel eScript languages.	None.
Syntax checking	Syntax checking.	None.
Declaration area	Declaration area.	None.

Table 53. Designing Scripts

Version 6.x	Version 7.0 and 7.5	Comment
Scripts can be written on product root	Scripts can be written on product root.	None.
Events return changes to categories	Events return changes only to items added from product table. Events do not return changes to relationship quantities.	Relationships are a grouping mechanism within a customizable product and are similar to categories.
Cfg_ItemInitialize	Use Cfg_InstInitialize.	Cfg_InstInitialize triggers once when session is started.
Cfg_ItemPreRequestSubmit	Not supported.	Can be simulated in some cases using the User Interface API.
Cfg_ItemChanged	Use this event only for child-products with components. Use Cfg_ChildItemChanged for other components.	None.
Cfg_CategoryChanged	Not supported.	None.
Cfg_SessionPostProcess	Not supported.	None.
Cfg_ItemPreSynchronize	Use Cfg_InstPostSynchronize.	None.
Cfg_ItemPostSynchronize	Cfg_InstPostSynchronize.	None.
Cfg_SessionClosed	Implemented at Instance Broker level. Use Siebel API.	None.
GetSessionId	GetInstanceId.	Returns the row ID of the customizable product in the source object, (quote, order and so on).
GetCfgId	Not supported. No longer meaningful.	None.
GetItemId	Not supported.	None.
GetItemQuantity	GetObjQuantity.	Returns the quantity of a component within the customizable product. Cannot be used for relationship quantities.
GetPropertyValue	Getting attribute value is supported through the Siebel API.	None.

Table 53. Designing Scripts

Version 6.x	Version 7.0 and 7.5	Comment
GetItemState	Supported through Siebel API	None.
SubmitRequest	AddItem, RemoveItem	None.

Quote Integration and Configuration Assistant in Version 6.x and 7.x

Creating a model product and associating a model with it is not required in version 7.0 and 7.5. Instead, you create a customizable product work space, which is similar to creating a record in the Model Manager in version 6.x. This work space associates the parts of the customizable product, including its components, links, resources, and UI definition with the product record. When you release a new version of the customizable product, it becomes available immediately for configuration in quotes and in eSales Web pages.

In version 6.x, the Configuration Assistant view was used to display models, select items, and transfer items to a quote. If you wanted to modify how the Configuration Assistant looked, you had to use Oracle's Web Tools to build a new view. In version 7.0 and 7.5, a Product UI Designer is provided within Siebel Configurator to create the browser pages, called selection pages, that will display during a configuration session. You can select from several base themes and product themes that define basic page layout. You can also select the controls, such as radio buttons or check boxes, to use for displaying items.

The base themes, product themes, and controls are stored in template files that you can customize or use to create your own templates. In addition, you can use the User Interface Property Designer to customize how individual items display. If you do not want to create a user interface for a customizable product, the application provides intelligent defaults for creating selection pages automatically. Table 54 maps version 6.x features to version 7.0 and 7.5 for quote integration and Configuration Assistant.

Table 54. Quote Integration and Configuration Assistant

Version 6.x	Version 7.0 and 7.5	Comment
Verify a solution	Verify a customizable product configuration.	None.
Verify a quote	Verify a quote.	None.
Update a quote	Update a quote.	None.
Solution name	Customizable product name.	None.
Solution quantity	Customizable product quantity.	None.
Line item quantity	Component quantity in a customizable product.	None.

Table 54. Quote Integration and Configuration Assistant

Version 6.x	Version 7.0 and 7.5	Comment
Reconfigure a solution with a new version of a model	Reconfigure a stored session with a newer version of a customizable product	None.
Create and manage Favorites	Create and manage Favorites	None.
Add an item in a configuration session	Same. Define item display in Product UI Designer	You can also accept application defaults for item display
Remove an item	Same. Define item display in Product UI Designer	You can also accept application defaults for item display
Add a category	Add a relationship in Structure view	None.
Unsatisfied category icon	A flag displays when a relationship is below its minimum cardinality or is required.	None.
Finish It!	Finish It!	None.
Item state explanation	The user interface contains an Explanation button.	Clicking the Explanation button displays an explanation.
Item properties displayed in applet	Properties have been replaced by configurable attributes. Attributes display in configuration session or in Quotes, then Dynamic Attributes.	None.
Messages and Recommendations	Messages and Recommendations	None.
Unsatisfied requirement message	Unsatisfied requirement message	None.
Quantity out of range message	Quantity out of range message	None.
Conflict-exists pop-up message	Conflict-exists pop-up message	Explanation and ability to proceed or cancel user action is supported
Edit quantity of an item	Edit quantity of an item	Requires UI control that allows editing of quantity.

Index

C

F

G

H

I

J

K

L

M

N

O

P

S

T

U