Oracle SCM Cloud Configurator Modeling Guide

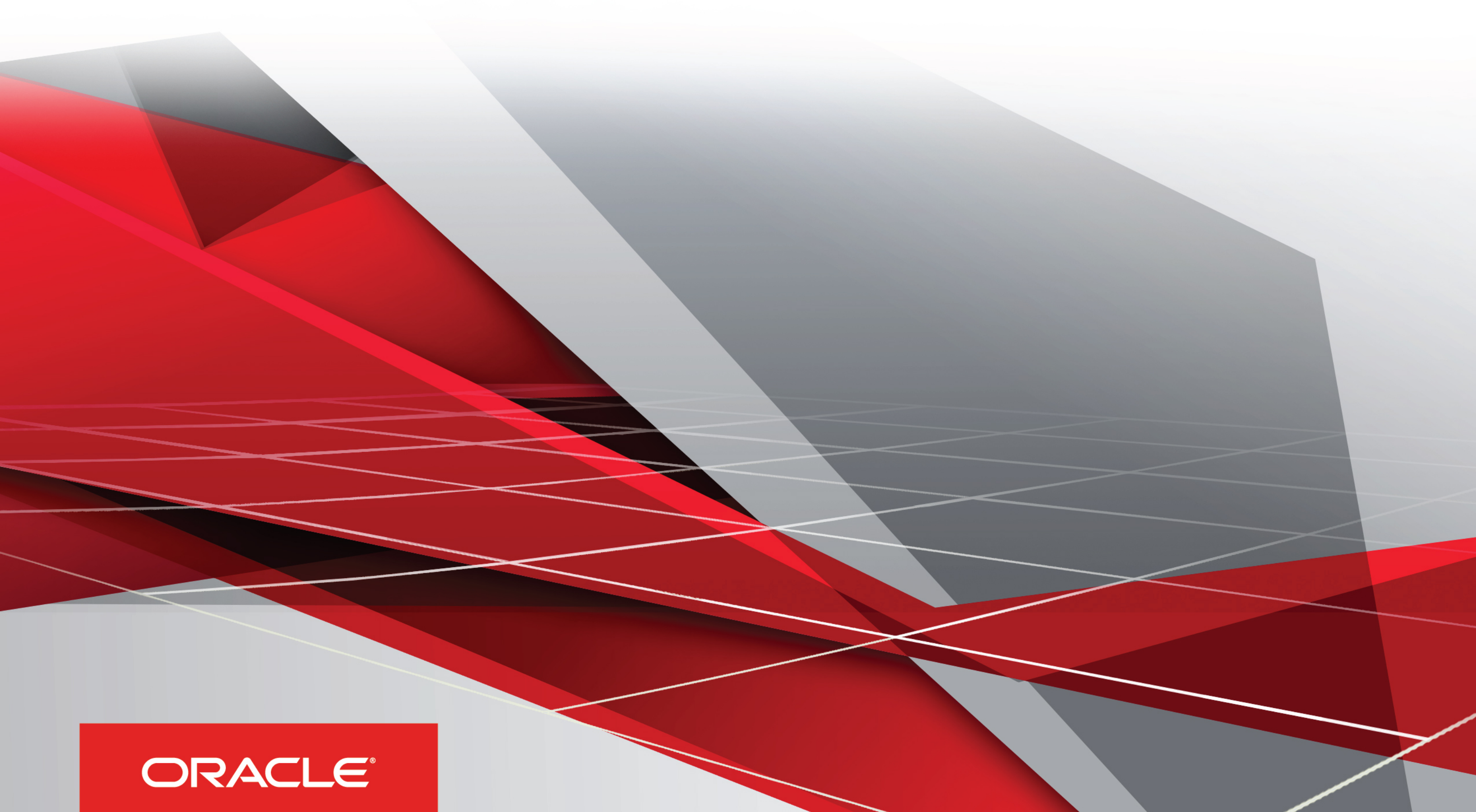
Oracle

SCM Cloud

Configurator Modeling Guide

Release 12

This guide also applies to on-premises implementations


ORACLE

Oracle® SCM Cloud Configurator Modeling Guide

Part Number E74112-04

Author: Mark Sawtelle

Contents

Preface

This preface introduces information sources that can help you use the application.

Oracle Applications Help

Use the help icon ? to access Oracle Applications Help in the application. If you don't see any help icons on your page, click the Show Help icon ? in the global header. Not all pages have help icons. You can also access Oracle Applications Help at https://fusionhelp.oracle.com.

Using Applications Help

Watch: This video tutorial shows you how to find help and use help features.

Additional Resources

- **Community:** Use **Oracle Applications Customer Connect** to get information from experts at Oracle, the partner community, and other users.
- **Guides and Videos:** Go to the **Oracle Help Center** to find guides and videos.
- **Training:** Take courses on Oracle Cloud from **Oracle University** .

Documentation Accessibility

For information about Oracle's commitment to accessibility, see the **Oracle Accessibility Program** .

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send e-mail to: oracle_fusion_applications_help_ww_grp@oracle.com.

1 Introduction to Configurator Modeling

The Configurator Models Work Area: Explained

The Configurator Models work area is the starting point for working with Oracle Fusion Configurator.

- Oracle Fusion Configurator functionality is based on configurator models.
- The Configurator Models work area is where you work with models based on snapshots, in workspaces.
- When you log in to Configurator Models, you always start on the Overview page.
- Since all changes to models are managed through workspaces that are in development status, the Overview page provides a filtered list of them, as a starting point.
- If your privileges allow it, you can create workspaces directly from the Overview page.
- If you don't want to start from the list of workspaces in development status, then use the tasks in the task list to manage objects directly.
- Dynamic tabs provide concurrent access to all major tasks.

Related Topics

- Using Snapshots and Models: Overview
- Using Workspaces: Overview
- Using Model Structure: Overview
- Using Model Rules: Overview
- Using Model User Interfaces: Overview

Configurator Model Creation and Maintenance: Explained

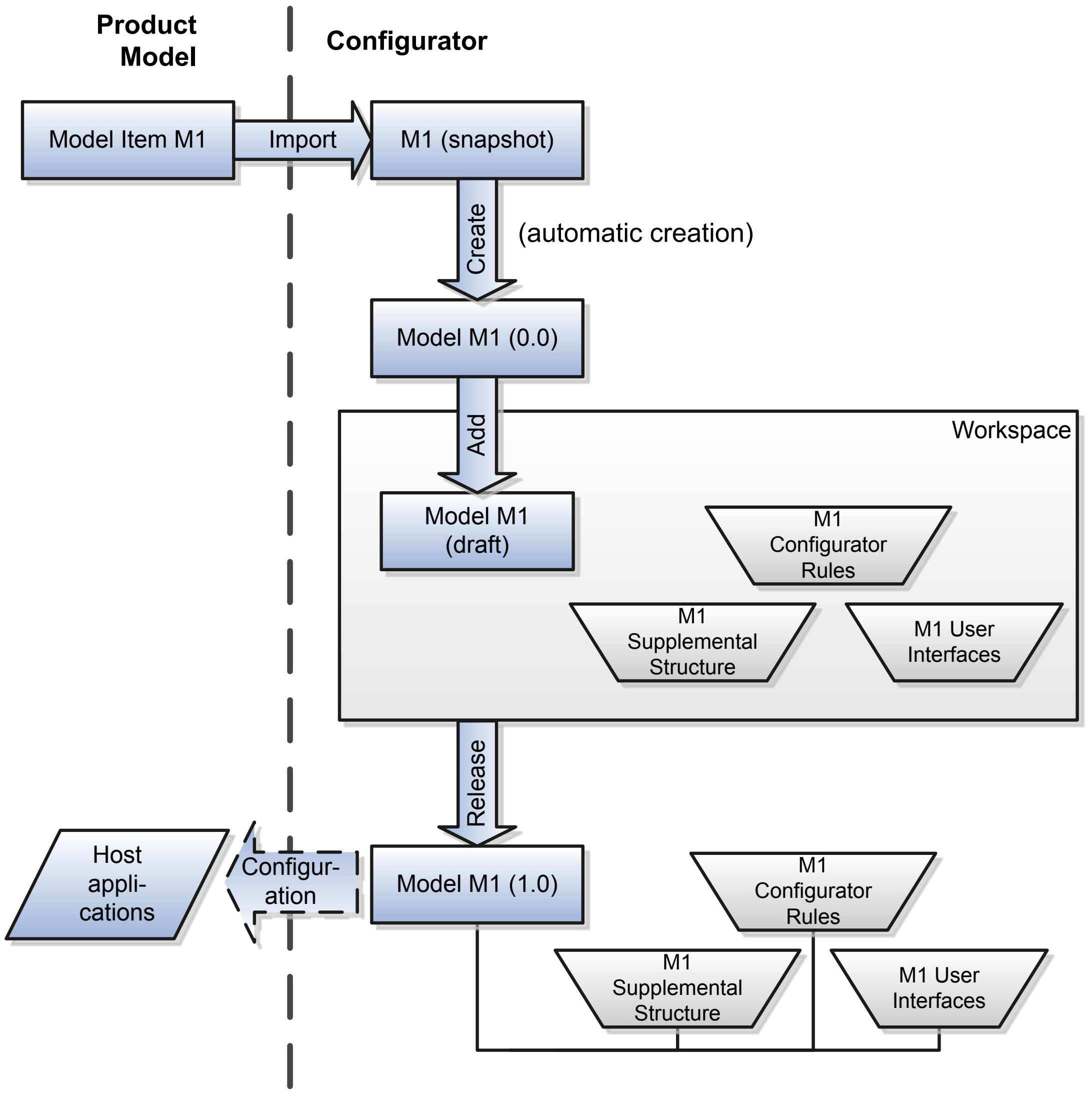
Configurator models are used by other applications to interact with configurable products and services. You create and maintain configurator models in the Configurator Models work area.

The phases of model creation and maintenance are:

- Bringing Model Data into the Configurator Models Work Area
- Making the Configurator Model Ready for Modification
- Modifying the Configurator Model Draft to Add Configuration Functionality
- Making the Configurator Model Available in Production
- Maintaining the Configurator Model

Note: Note: The procedure presented here is generalized for clarity, and may not incorporate all of the actions you need to perform.

The following diagram shows how a product item is imported as a snapshot and becomes a configurator model that is modified in a workspace, which is released to produce a new version of the model that provides configuration functionality to host applications.



Bringing Model Data into the Configurator Models Work Area

To create a configurator model, you must first import a product model item from Oracle Fusion Product Hub.

- On the Manage Snapshots page, use Import Model Item from the Actions menu to start the import process. After selecting a product model to import, submit the import, which creates a scheduled process.
- If the scheduled import process is successful, it creates a snapshot of the product model's data. The new snapshot can then be searched for on the Manage Snapshots page.

- The import process automatically creates a configurator model from the snapshot. The new model can then be searched for on the on the Manage Models page. The new model is the baseline for all further draft modifications and versions of the model.

Making the Configurator Model Ready for Modification

Before you can modify a configurator model to add configuration functionality, you must add the model to a workspace.

- On the Manage Workspaces page, create a workspace, to contain the additions and changes you intend to make to one or more configurator models.
- When you create a workspace, you must set an effective start date for that workspace. The effective start date is the date on which the changes to any objects in the workspace take effect. You can change the effective start date until you release the workspace.
- After you create the workspace, select and edit it. While you're editing a workspace, its Status is In development.
- On the Workspace Attributes and Participants page, add the configurator model that you want to modify to the workspace, by selecting **Select and Add Models** from the Actions menu.
- In the context of this workspace, whose status is **In development**, the model is considered a draft of the next version of the model.

Modifying the Configurator Model Draft to Add Configuration Functionality

While a model is a draft, you can add supplemental structure, model rules, and user interfaces that enhance the ability of end users to configure the product item.

- All modifications to a model are performed by editing a draft of the model in a workspace. On the Workspace page, select and edit a participating model.
- On the Structure tab of the Edit Configurator Model page, you can create model structure that supplements the imported configurable structure of a model item.
- On the Rules tab of the Edit Configurator Model page, you can create configurator rules in addition to any rules implicit in the imported model item.
- On the User Interfaces tab of the Edit Configurator Model page, you can create user interfaces in addition to the default user interface provided by Configurator.
- You can test the behavior of the model draft at any time while you're editing it, by clicking the Test Model button on the Edit Configurator Model page, which launches a test session in a new **Test Model** tab, with test parameters that you specify.

Making the Configurator Model Available in Production

To put your workspace modifications to the model draft into production, you must release the workspace. The modifications take effect upon the effective start date of the workspace.

- After editing and testing your model, return to the Workspace Attributes and Participants page.
- Ensure that the effective start date of the workspace is correct. You can change the date until you release the workspace.
- Click the Release button to release the workspace, which creates a scheduled process for the release.
- If the scheduled import process is successful, it creates a new version of the model. The new version appears on the Manage Models page, with an incremented version number.
- The new version of the model becomes the effective version of the model in production. The new version also becomes the baseline for further draft modifications of the model.

Maintaining the Configurator Model

After a configurator model is released, you can update it to further modify its configuration functionality, or to reflect changes in the underlying product item model.

- To add new configuration functionality for a model, create a new workspace, on the Manage Workspaces page, then add the model to it, as a new draft. In the new workspace, add new supplemental structure, configurator rules, or user interfaces, test the model, then release the workspace, producing a new version of the model.
- To reflect changes in the underlying product item model, add the corresponding configurator model to a workspace. On the Workspace Attributes and Participants page, select **Add Updated Item Snapshots for Models** from the Actions menu. The snapshot is added to the workspace, and the configurator model structure reflects changes to the product item. The model changes produce a new model version when you release the workspace.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together
- Configurator Models: Explained
- Item-Based Model Structure: Explained
- Creating Statement Rules: Explained
- Configurator Model User Interfaces: Explained

2 Using Snapshots and Models

Using Snapshots and Models: Overview

This chapter covers snapshots and models. Snapshots are imports of product data from which configurator models are created. The snapshots capture the product data as of the time of the import.

Snapshots are read-only definitions of approved product data imported from the Product Information Management work area into the Configurator Models work area.

You access snapshots and models in the following ways.

- **Configurator Models work area - tasks panel tab - Manage Snapshots**

  The Manage Snapshots page allows you to search for snapshots, by name, description, status, snapshot type, or structure item type. You can import snapshots of new product data or refresh the product data for selected snapshots.
- **Configurator Models work area - tasks panel tab - Manage Models**

  The Manage Models page allows you to search for models, by the name or description of an original product item or of the model created from that item. You can open any of the drafts or released versions of a selected model.

Snapshots: Explained

Snapshots are imports, into the Configurator Models work area, of product data maintained in Oracle Fusion Product Hub, which is accessed through the Product Information Management work area. Snapshots copy production data at a point in time. Each snapshot of a model item corresponds to a configurator model that is created from it. Snapshots in Released status are the source of the product data for configurator models released into production. When a model item is imported as a snapshot, an item-based configurator model is automatically created. After import, you can refresh snapshots to maintain the synchronization of an item-based model with changes to its original source item in Product Hub.

Aspects of snapshots include:

- Creating Snapshots
- Purpose of Snapshots
- Updating Snapshots
- Sharing Snapshots
- Capturing Data in Snapshots
- Snapshots and Versions

Creating Snapshots

A new discrete snapshot is created for each product item by the Import operation, resulting in a snapshot status of **Released**. Snapshots can be updated by the Refresh operation, resulting in a snapshot status of **Modified**.

Snapshots of item classes and value sets associated with item components in a model's structure are imported when the model is imported for the first time. The important data associated with an item class is the set of transactional item attributes (TIAs) defined on it. The value sets that are imported are those that are associated with the TIAs in the imported item classes. When item classes or value sets are updated in the Product Information Management work area, you must explicitly refresh their snapshots in the Configurator Models work area to obtain the updates. These snapshots are not automatically updated when the associated model is refreshed.

Important: To keep transactional item attribute values current in your configurator models, you must refresh the item class and value set snapshots associated with the model.

Updating Snapshots

When the source product data item in Product Hub changes, you can refresh the snapshot corresponding to that item, if a business decision requires doing so. Configurator then sets the status of the snapshot to Modified, indicating a possible need to update the configurator models built on that snapshot data. To update model drafts using the modified product data in the snapshot, you must add the modified snapshot to the workspaces containing the model drafts. When the model updates are complete and tested, and you release the workspace into production, then the snapshot status changes from Modified to Released. Remember that item class and value set snapshots must be explicitly refreshed.

Sharing Snapshots

Configurator shares the same single snapshot among all models that reference that product item.

The data changes produced by refreshing are applied to any other configurator model drafts that use the snapshot, after the updated snapshot is added to a workspace and the workspace is successfully released.

Important: When you refresh a snapshot, any updated data potentially affects every configurator model that is based on that snapshot. You should test every model affected by snapshot refreshes before releasing it into production.

Capturing Data in Snapshots

Snapshots capture a selected subset of the data in a product item.

The following table describes the elements of data included in a snapshot in Configurator. These elements are available for display as columns for the snapshots listed on the Manage Snapshots page.

Name	Data Type	Source of Data	Description
Name	Text	Imported from Product Hub	Item name or number
Description	Text	Imported from Product Hub	Item description
Organization	Text	Imported from Product Hub	Item organization

Name	Data Type	Source of Data	Description
Status	Code	Created by Import operation	Values are: • **Released**: indicates a Snapshot being used by run time Configurator models • **Modified**: indicates a successful refresh with changes detected. If refresh finds no changes, status reverts to Released. If refresh fails, status reverts to Released.
Snapshot Type	Code	Created by Import operation	Values are: • Item • Item Class • Value Set
Structure Item Type	Code	Imported from Product Hub	The type of the imported model item. Values are: • Standard Item • Option Class • Model
Last Refreshed By	Text	Created by Refresh operation	Last user who refreshed the snapshot.
Last Refresh Date	Date/Time	Created by Refresh operation	Date when any aspect of the snapshot was last updated by a Refresh operation.
Last Released By	Text	Created by Release operation	Last user who released a workspace containing the snapshot.
Last Release Date	Date/Time	Created by Release operation	Date when any aspect of the snapshot was last updated via the Release operation.

Snapshots and Versions

Although a snapshot is associated with a configurator model, a snapshot does not have versions, as a model does, because a snapshot cannot be modified. A snapshot is a reflection of the state of a product item at the moment it is imported into the Configurator Models work area. Changes made to item versions or revisions in the Product Information Management work area only affect snapshots if they change the attributes that are imported in a snapshot.

- When a model item is first imported from Product Information Management into the Configurator Models work area, a baseline version of the snapshot of that model item is released.

- The baseline version is also referred to as version zero, because its version number is 0.0.
- The baseline versions of all the constituents of the imported model item (option classes, standard items, reference models, item classes, and value sets) are also released.
- If the model is configured from a transactional application before a completely developed configurator model goes into production via a workspace release, then the baseline version of the model is used instead.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together

Importing Items: Explained

You import items from Oracle Fusion Product Hub that have a Structure Item Type of Model to create snapshots from which item-based configurator models are created. When you import an item, Oracle Fusion Configurator copies selected elements of this item data and stores them in a snapshot of the selected item at the time of the import.

Aspects of importing items include:

- Importing Items
- Selecting Items to Import
- Results After Import
- Making User-Defined Attributes Available for Import

Importing Items

You import items on the Manage Snapshots page.

On the Manage Snapshots page, choose the Import Model Item command. In the Search and Select: Product Model Item dialog, search for the item that you want to import.

> **Important:** You cannot import the same item more than once. The list of available items excludes items that have already been imported as snapshots. To obtain the latest updates to an item you must refresh its snapshot.

When the search results include the desired item, select that item and click the **Submit** button. A scheduled process is submitted, and a confirmation message provides you with the process request ID that you can use to check on the status of the Import Product Model Item process on the Scheduled Processes page.

After the process completes successfully, search on the Manage Snapshots page for the item that you imported. The item now appears as a new snapshot in the search results, along with new snapshots for any required child items. If the children are shared among other models that have previously been imported, the child items are not re-imported. However, if the child items had been previously imported and have had subsequent updates, those changes will be refreshed as part of the parent model import. The status of those updated child snapshots will be Modified.

Selecting Items to Import

You can refine your search for items to import, to help you find the exact item desired.

Any ATO or PTO model that has optional structure can be imported. Any model item for which Primary structure has been defined can also be imported. The suitability of a product model item is not dependent on the presence or value of a particular attribute.

When you select items to import, in the Search and Select: Product Model Item dialog, you can use the advanced set of search fields, which enable you to include item catalogs and categories in your search. Click the **Advanced** button to display the advanced search fields.

You can refine the search results by adding columns that display more of the large number of attributes associated with items. Select **Add Columns** from the View menu, then search for and select available attributes in the Add Columns dialog. You can query by example in the list of available attributes, then select attributes and click the **Add** button. The selected attributes are displayed in the dialog, and are included in the search results for items to import.

Important: Adding attributes to the search results does not add them to the set of attributes that are imported.

Results After Import

Importing an item produces a snapshot of the item.

In addition to a snapshot of the item itself, import produces snapshots of:

- The item's item class, if a snapshot of that item class doesn't already exist
- Any parent item classes of the item's item class, if they don't already exist
- The value sets associated with the transactional item attributes defined on the item's item class

The imported item's name or number becomes the Name of the snapshot, and the imported item's description becomes the Description of the snapshot. The following attributes are imported:

- Item attributes:
    - Description

      Item Type

      Primary UOM Code

      Eligibility Rule (called Eligibility Enabled after import)

      Indivisible Flag

      Organization

      Serial Generation

      Track in Installed Base
- Component attributes
    - Minimum Quantity

      Maximum Quantity

      Default Quantity

      Optional Children Are Mutually Exclusive

Required When Parent Is Selected

Start Date

End date

Sequence Number

Instantiation Type

Show in Sales

By default, snapshots include only those user-defined attributes that:

- Have been associated to the Configurator Functional Area via an item page on the item class tied to the item, in Product Hub
- Currently have values
- Are defined at the item level

When a model item snapshot is created by import, its snapshot status is automatically set to **Released**, and a corresponding configurator model is created and released. The new model version has the version number 0.0 (zero), and is referred to as the zeroth version. This version is the baseline for all further draft modifications and versions of the model. The zeroth model is automatically added to a workspace that cannot be edited, which is named with the form `+<snapshot name> Model Creation Workspace`.

Making User-Defined Attributes Available for Import

Importing an item produces a snapshot of the item.

By default, the snapshot definition of an item will include only those user-defined attributes that have been associated to the Configurator Functional Area via an item page on the item class tied to the item in Oracle Fusion Product Hub. You may associate attribute groups in various item classes to the item page associated to the Configurator Functional Area to expose different user defined attributes on different items.

Importing Transactional Item Attributes

You make transactional item attributes (TIAs) for an item available to your configurator model by importing, and later refreshing, snapshots of item classes and value sets.

The definition of a TIA for an item is part of the item class from which the item was created in the Product Information Management work area. You provide this definition to your configurator model by importing a snapshot of the item class, and adding that snapshot to the workspace in which you work on your draft modifications to the model. You keep the TIA definition current relative to its original in the Product Information Management work area by refreshing the item class snapshot.

The values for the TIA that are presented to an end user at run time are defined in a value set in the Product Information Management work area. As with item classes, you provide this set of values to your configurator model by importing a snapshot of the value set, and adding that snapshot to the workspace in which you work on your draft modifications to the model. As with item class snapshots, you keep the values current relative to the Product Information Management work area by refreshing the value snapshot. At run time, the end user provides TIA values by selecting an edit control on the item, then selecting or entering the value in a dialog box for the TIA.

There are some restrictions on using TIAs with Configurator:

- Date and Date/Time types for TIAs aren't supported.
- In the definition of a TIA in the item class hierarchy in the Product Information Management work area, modifications associated with child item classes may further constrain the definition of the TIA, but may not relax an inherited definition.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together

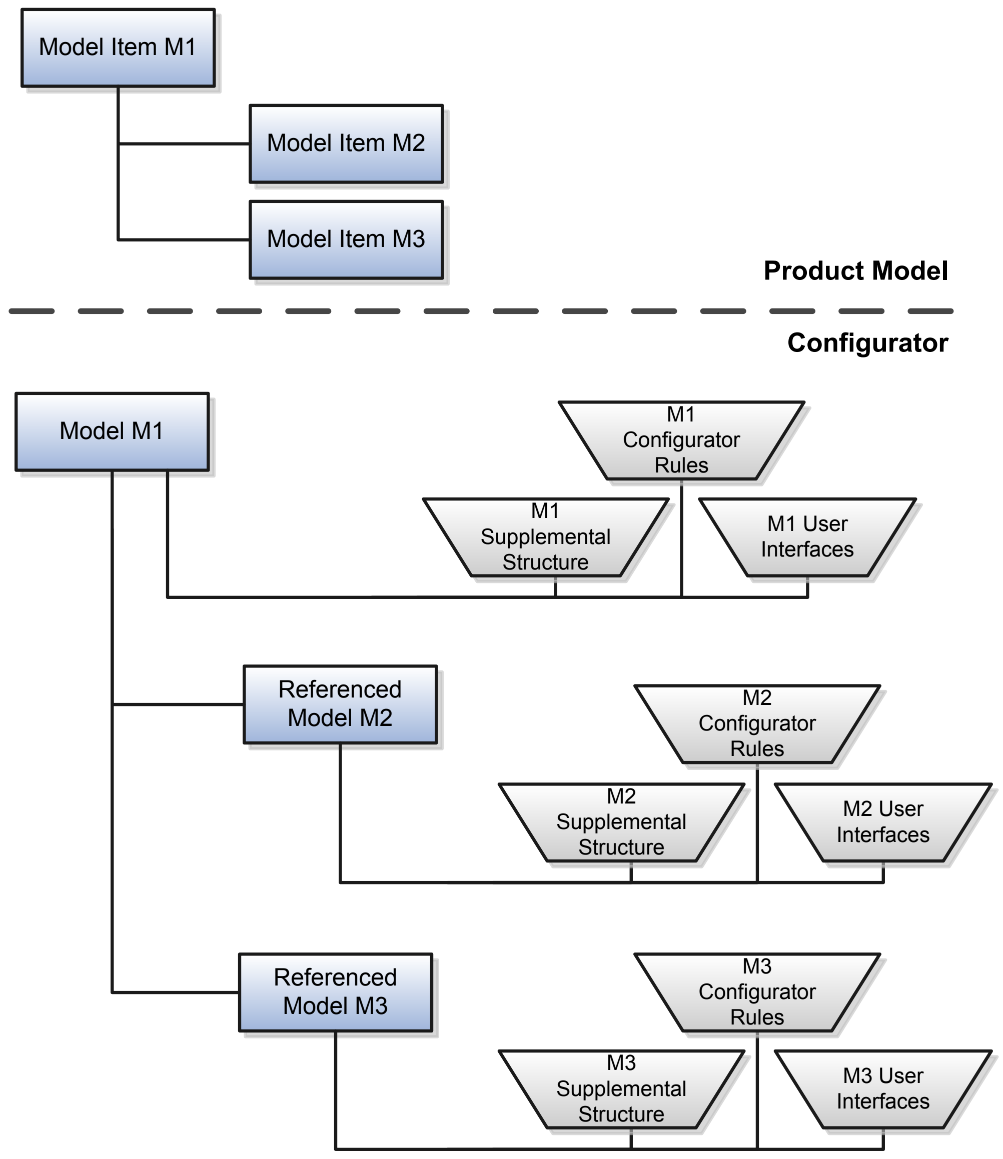
Configurator Models: Explained

Configurator models are the means by which end users configure products and services. Models incorporate the structure of the item to be configured, the configurator rules that you create to define configuration behavior, and optional user interfaces that enhance the end user's configuration experience.

Aspects of configurator models include:

- Model Structure
- Referenced Models
- Configurator Rules
- User Interfaces
- Model Versions

The following figure shows the elements of a configurator model, and that the structure of a configurator model is based on the structure of a model item in the Product Information Management work area.



Model Structure

You can add supplemental structure to the imported model structure that is the foundation for the configurator model.

Configurator models are imported from items maintained in Oracle Fusion Product Hub (which is accessed by the Product Information Management work area) that have a Structure Type of Model. Only the optional structure and attributes of the product item are imported, because the required elements of the product item cannot be configured by an end user. The option classes and optional standard items for each model item are also imported.

When an end user interacts with a configurator model during a configuration session, he or she selects from or provides values for certain fields and attributes.

The imported model structure is represented in the Configurator Models work area as an expandable hierarchical tree, in the Structure Hierarchy pane of the Structure tab of the Configurator Model page. Each model, option class, or standard item is a node in the tree.

You can add supplemental structure to the imported model structure. You can use supplemental structure to represent guided selling questions that solicit inputs from the end user, or to store inputs and selections from the end user that are used in configurator rules. By adding the available types of supplemental structure feature nodes to the structure tree, you can add lists of options, integer and decimal inputs, and Boolean choices.

You create supplemental structure on the Structure tab of the Edit Configurator Model page. Before you can add supplemental structure, you must add the model to a workspace, as a draft.

Referenced Models

Model items in the Product Information Management work area can have child models, which can in turn have child models. Such child models are imported as Configurator nodes called model references.

In the Product Information Management work area, before import, you can define the number of instances that can be created of a model reference during a run time configuration session. The attribute that defines this setting is called instantiability.

Referenced models have their own supplemental structure, configurator rules, and user interfaces, independent of those belonging to their parent models.

Configurator Rules

You can add configurator rules, which define how separate parts of a model are related to one another and ensure that users will make valid selections when configuring the model. You can write rules between nodes of a model, including nodes in child referenced models.

You define configurator rules by creating and editing statement rules, which are written in the Constraint Definition Language (CDL). You create configurator rules on the Rules tab of the Edit Configurator Model page. Before you can create rules, you must add the model to a workspace, as a draft.

The structure imported from the Product Information Management work area to the Configurator Models work area incorporates some implicit functionality that is similar to rules. For example, some option classes are defined to be required when their parent is selected, or have child options that are mutually exclusive.

User Interfaces

You can add user interfaces (UIs), to enhance the end user's configuration session experience.

When an end user interacts with a configurator model during a configuration session, the model is presented using one of a possible set of UIs. A default UI is provided for every model, displaying all of the nodes of the model on a single page. You can create alternative UIs that suit the particular needs of your intended end users.

You define UIs on the User Interfaces tab of the Edit Configurator Model page. Before you can create UIs, you must add the model to a workspace, as a draft. When you create a UI, it is generated automatically by applying a template that you select to the model structure.

Note: Note: Each configurator model has its own supplemental structure, configurator rules, and user interfaces. These elements of the model are not inherited by or shared with other models.

Model Versions

Configurator models are versioned objects. A model can go through multiple drafts before being released in a series of versions.

Every modification to a model that goes into production must be part of a workspace that is released. After the workspace is released, the version number of the model is incremented to the next available number for that model. When the effective start date of the workspace arrives, the model changes become effective.

There can be multiple drafts of the same model being modified at the same time, but they must be in different workspaces. A message notifies you if you add a model to a workspace when another draft exists in another workspace. There can only be a single released version of a model with the same effective start date.

You can view all of the current drafts and release versions of a model by searching for the model on the Manage Models pages and selecting it in the search results.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together
- Configurator Model Creation and Maintenance: Explained
- Item-Based Model Structure: Explained
- Configurator Rules: Explained
- Configurator Model User Interfaces: Explained

Refreshing Snapshots: Explained

You refresh snapshots in order to keep your configurator models synchronized with changes to production items.

Refreshing Snapshots

To refresh a snapshot:

1. Navigate to the **Manage Snapshots** page.
2. Search for and select the snapshot to be refreshed. You can refresh snapshots for:
    - Items (models, option classes, and standard items)
    - Item classes
    - Value sets

3. From the **Actions** menu, select the appropriate refresh action.

   - Refresh
   - Refresh Including Descendant Structure
   - Refresh Including Descendant Structure and Referenced Structure

   The refresh actions are described under Refresh Depth.
4. The selected Refresh action submits a scheduled process, notifying you of the request ID number. You can search for the process and view its log in the Scheduled Processes work area. The process name is Refresh Snapshot.
5. Updates to existing items that are brought into the Configurator Models work area by the refresh process are shown with the status **Modified**. After the workspace is released, the snapshot status is changed from Modified to Released. New items that are brought in by the refresh process are immediately released to production, and have the status **Released**.

Refresh Depth

You can selectively refresh snapshots, choosing an action that determines how deeply into the item structure any product updates will be reflected.

The actions for refresh depth are described in the following table.

Refresh Depth	Effects
Refresh	Brings all approved changes to the selected item into the snapshot in the Configurator Models work area, including the additional items that are added to the item's structure. Changes to the model item, are restricted to: • Operational attributes of the item itself. • Immediate item structure, meaning the immediate child items of a model item, or options of an option class. • Operational attributes of the child items or options.
Refresh Including Descendant Structure	Same as above, but also extends through any descendant structure, such as an option class under an option class.
Refresh Including Descendant Structure and Referenced Structure	Same as above, but also extends through any descendant referenced structure, such referenced model items.

Example of Refresh

The purpose and method of refreshing snapshots is illustrated by the following typical scenario.

1. A model item M1, maintained in Oracle Fusion Product Hub, in the Product Information Management work area, has previously been imported into the Configurator Models work area, creating a configurator model M1 that is subsequently released as a model version.
2. In the Product Information Management work area, a product manager adds a new standard item SI4 to an option class OC2 in the production model M1.
3. The snapshot of the model item M1 in Configurator is now out of date. The new item SI4 will not appear if the model is used in a configuration session.
4. You refresh the snapshot of M1 on the Manage Snapshots page of the Configurator Models work area, using the **Refresh Including Descendant Structure** action.

5. After refresh, the Status of the snapshot of the option class OC2 is now **Modified**, and it now includes the new standard item SI4 for the option class OC2.

   An item snapshot definition is similar to a item definition. The snapshot includes the item and its direct components. Components deeper in the structure belong to the item snapshots of their parent items.
6. You add the snapshot to a new workspace containing a draft of the configurator model. The snapshot imparts the latest product data to the model, including the addition of item SI4.
7. You possibly make changes to the model, using the product changes derived from the snapshot.
8. You test the configurator model, to determine whether your changes to the model draft produce the desired configuration behavior.
9. If testing is successful, you release the workspace, which makes the model draft into a new production version, and changes the status of the snapshot to **Released**

   > **Important:** When you refresh a snapshot, any updated data potentially affects every configurator model that is based on that snapshot. You should test every model affected by snapshot refreshes before releasing it into production.

10. After the snapshot status becomes Released, its product data updates are reflected in all other models that use that snapshot.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together
- Configurator Model Creation and Maintenance: Explained
- Item-Based Model Structure: Explained
- Configurator Rules: Explained
- Configurator Model User Interfaces: Explained

FAQ for Using Snapshots and Models

What's a snapshot?

A snapshot is an imported copy of the latest configurable data in a product model item that is created and refreshed in the Configurator Models work area. Configurator models are based on the item definition imported as a snapshot.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together
- Configurator Model Creation and Maintenance: Explained

How can I create a configurator model?

When you import a model item, a configurator model is created automatically, along with a snapshot.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together
- Configurator Model Creation and Maintenance: Explained

Why don't I see any standard items in the model or its option classes, after importing a model?

To be configurable, child items must be marked as optional in the Product Information Management work area. Otherwise, those items will not be imported, and will be absent from the imported model.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together
- Configurator Model Creation and Maintenance: Explained

Why don't I see any recent changes to my original item reflected in my configurator model?

The corresponding item snapshot needs to be refreshed, and added to the workspace where the configurator model is a participant.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together

What's the difference between a product model item and a configurator model?

A product model item includes a full set of structure and attributes that describe the item and enable related processes in which it is used.

A configurator model has only a subset of the structure and attributes of a model item, namely the optional structure that can be configured. A configurator model can also contain supplemental structure, rules, and customized user interfaces.

Related Topics

- Workspaces, Snapshots, and Models: How They Work Together

- Configurator Model Creation and Maintenance: Explained

Why did a model open read-only?

You can only modify model versions that are drafts, and that you have locked for editing. If a model opens read-only, it may be a released version, or a draft locked by another user.

Related Topics

- Versions: Explained
- Workspaces, Snapshots, and Models: How They Work Together

3 Using Workspaces

Using Workspaces: Overview

This chapter covers workspaces. Workspaces are containers in which you edit and test drafts of changes that you make to configurator models. After you release a workspace, changes to drafts become effective on the effective start date that you define for the workspace.

- Workspaces enable you to modify and test one or more draft models prior to release into production.
- Modified snapshots can also be added to workspaces along with models.
- Multiple workspaces allow concurrent draft development.
- After testing, workspaces can be released so that their contents are available in production. All changes made to the models are effective as of the effective start date of the workspace.

You access workspaces in the following ways.

- **Configurator Models work area - Overview page**

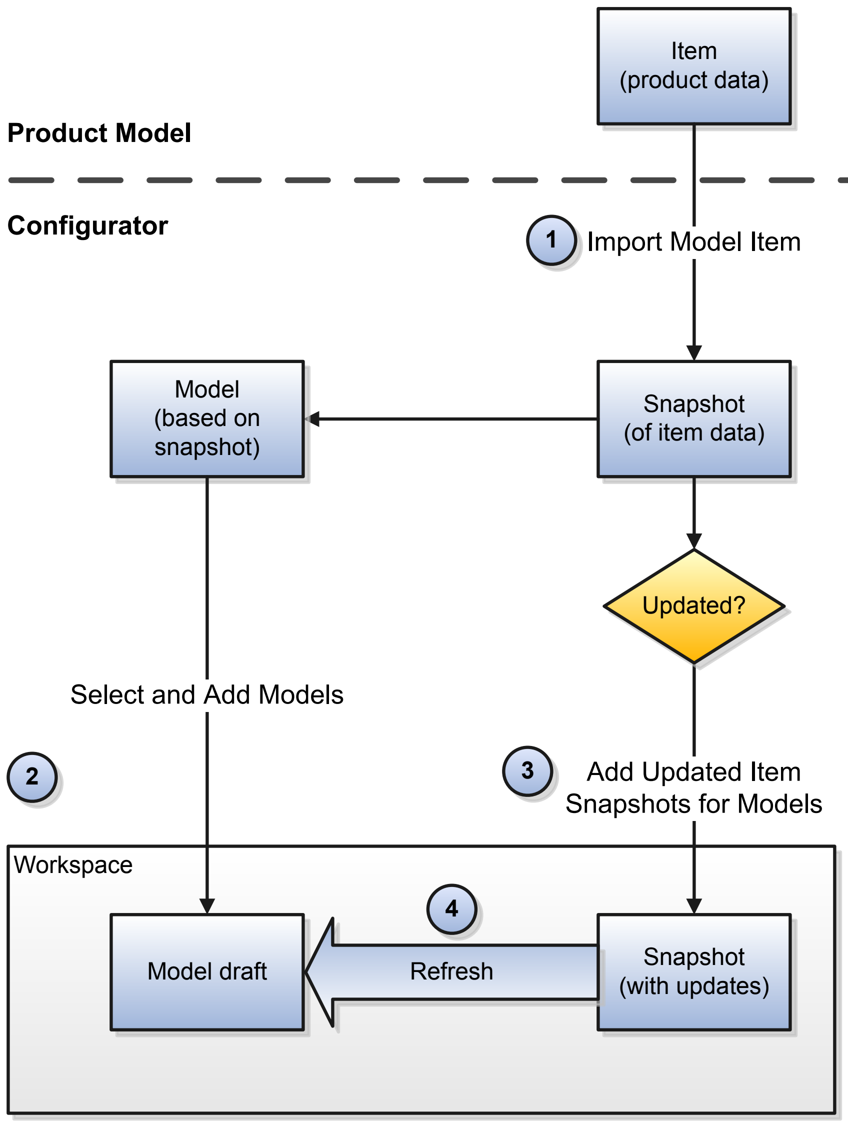
  The **Workspaces in Development** table lists workspaces whose status is **In Development**. You can filter the view to show all workspaces or only those belonging to the current user.
- **Configurator Models work area - tasks panel tab - Manage Workspaces**

  The Manage Workspaces page allows you to search for workspaces, by name, description, status, or creator. You can access released workspaces as well as those in development.

Workspaces, Snapshots, and Models: How They Work Together

Configurator models are created from snapshots of product item data. Snapshots added to a workspace keep model drafts in that workspace current with item updates when the snapshots are refreshed

The following diagram shows that item data is imported as a snapshot, is used to create a configurator model, and can be refreshed so that product updates are reflected in a workspace.



Role of Snapshots

Each item-based configurator model is created by importing a subset of the product data for that item. Only the item data that is used by Oracle Fusion Configurator is imported, including user-defined attributes. The imported data is called a snapshot, because it captures the item data at the time of the import.

Configurator maintains the relationship between the snapshot and the original item. If the original item is updated in Oracle Fusion Product Hub, the updates are reflected in the snapshot when you select the snapshot on the Manage Snapshots page and choose the **Refresh** command.

Role of Models

When a product item is imported as a snapshot, an initial version of the item-based configurator model is automatically created from the imported data. Modifications are not allowed to this initial model version. Modifications to a configurator model must be made within a workspace, where you create a draft of the model based on the initial model version or the most recently released version by adding the model as a participant in the workspace. After you release the workspace, the draft becomes an effective version of the model upon the effective start date that you specified for the workspace.

Role of Workspaces

Workspaces contain drafts of new versions of configurator models. The effective start date for these versions is an attribute of the workspace, which you define when you create the workspace. You add a model to a workspace as a participant. You can add a model by itself, or with its referenced child models.

Workspaces can also contain any refreshed snapshots that correspond to the models in the workspace. When such a supporting snapshot is added to the workspace, any updates to the snapshot are reflected in the model that was created from that snapshot. To add a snapshot to a workspace, select the corresponding model on the Workspace page and select the **Add Updated Item Snapshots for Models** command.

- You can add an updated snapshot when you add a model to a workspace, by selecting the **Include updated item snapshots for models** check box on the Select and Add: Models dialog.
- You can't add a snapshot unless it has first been updated by the Refresh operation.

Related Topics

- Configurator Model Creation and Maintenance: Explained
- Snapshots: Explained
- Configurator Models: Explained

Workspaces: Explained

A workspace is a container in the Configurator Models work area in which you edit and test drafts of changes that you make to configurator models. After you release a workspace, changes to drafts become effective on the effective start date that you define for the workspace. A workspace is the place where you create all modifications to a configurator model. You modify a model by adding configurator rules, supplemental structure, or user interfaces. A configurator model with a set of modifications made in a specific workspace is called a draft of that model.

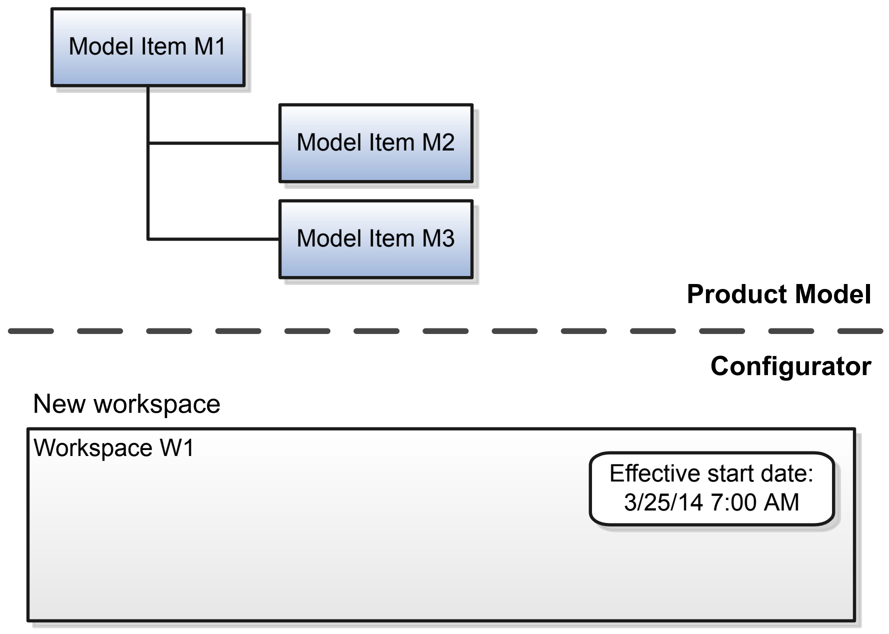
Aspects of workspaces include:

- Creating Workspaces
- Adding Participants to Workspaces
- Locking and Unlocking Participants
- Status of Workspaces
- Cautions about Concurrent Development of Participants
- Releasing Workspaces

Creating Workspaces

You create workspaces on the Manage Workspaces page, or on the Overview page, in the list of workspaces in development. When you create a workspace you must specify a unique name and an effective start date, which is the date on which your modifications to model drafts go into effect. You can edit the workspace attributes, including its name and description.

The following figure shows a model item in Oracle Fusion Product Hub, and a new workspace in Oracle Fusion Configurator that does not yet contain any draft models to be modified. An effective start date must be specified whenever you create a workspace.



Note: You cannot delete workspaces.

Adding Participants to Workspaces

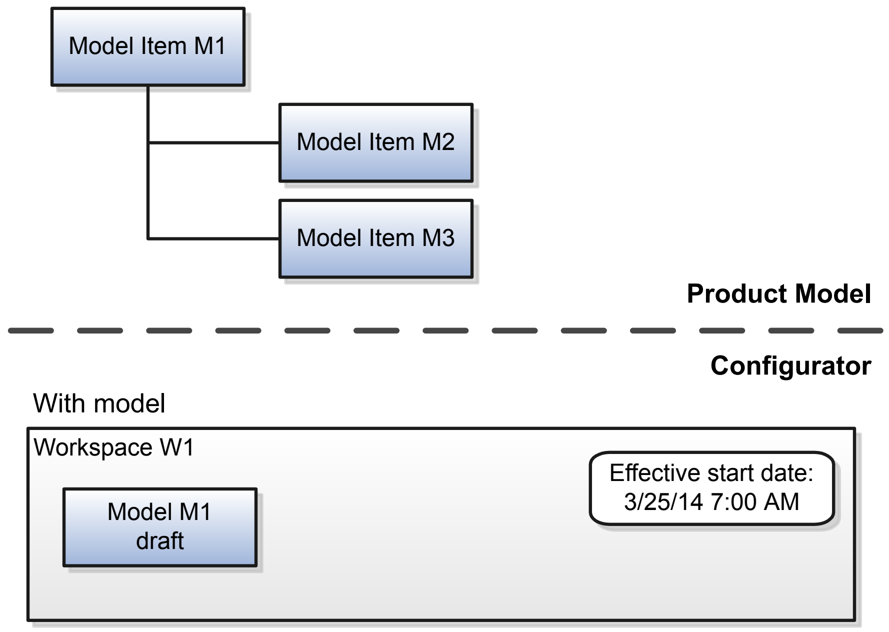
Before you can modify a model, you must add it to a workspace, thus creating a draft of that model. In the workspace, you can then make changes to the model draft. When an object is added to a workspace it is called a participant of that workspace.

- You cannot modify a model except as a draft in a workspace.
- If you remove a draft from a workspace, all the changes made to the draft are permanently lost.

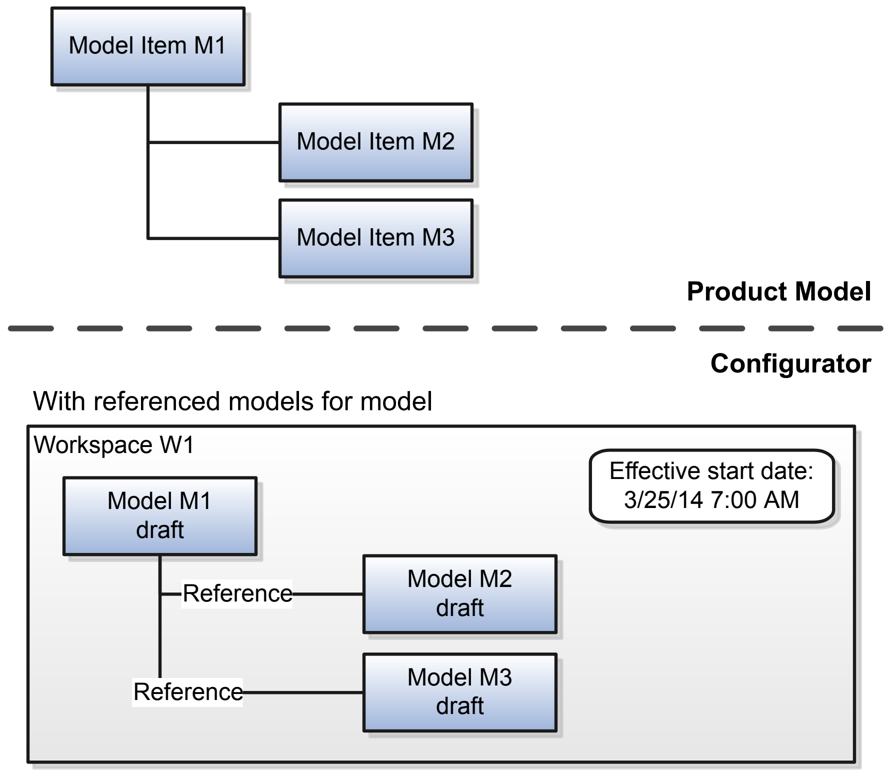
You can add the following objects to workspaces as participants:

- Models: You can select and add configurator models from a list of model items that have already been imported as snapshots.
- Referenced models: You can add model's child referenced models when you add the model, or later.
- Snapshots: You can include a model's updated snapshots when you add it, or later.

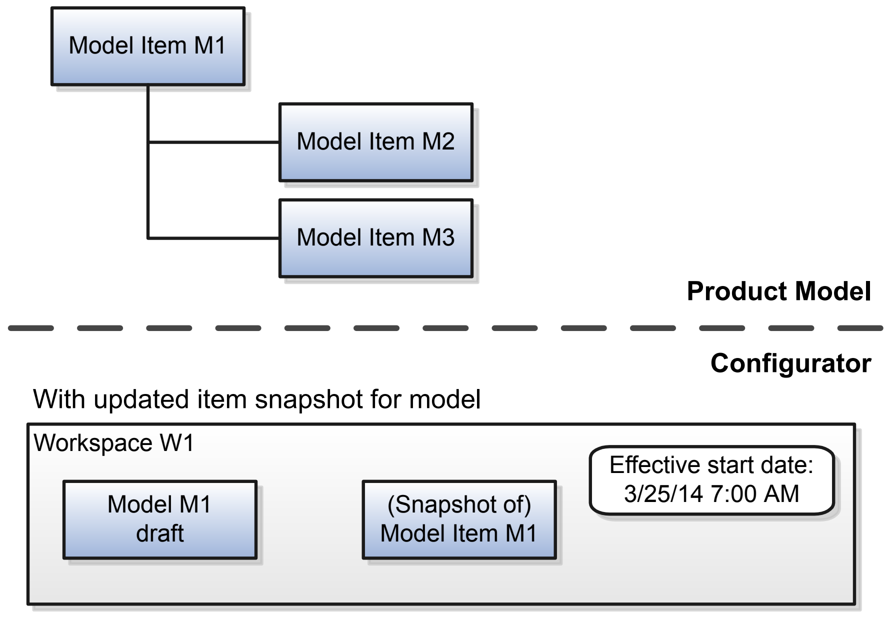
The following figure shows a model item added to a workspace, without its child referenced models.



The following figure shows a model item added to a workspace, with its child referenced models.



The following figure shows a model item added to a workspace, with its updated snapshot. You would add a snapshot if it had been updated by the Refresh operation, so that the updates would be applied to the corresponding model as part of the changes in the workspace.


Model Item M1
Model Item M2
Model Item M3
Product Model
Configurator
With updated item snapshot for model
Workspace W1
Model M1 draft
(Snapshot of) Model Item M1
Effective start date: 3/25/14 7:00 AM

Locking and Unlocking Participants

To prevent making changes that conflict with another user's changes to the same object, in the same workspace, you must lock a participant before you edit it. You lock participants on the Workspace page, by selecting them in the list of workspace participants and clicking the Lock button. When you add a participant, it is automatically locked for you.

- While participants in a workspace can be locked, workspaces are not lockable. Any number of users can edit participants concurrently.
- Locking prevents other users from changing a draft in the same workspace, but does not prevent changes to a draft of the same object in a different workspace.

Status of Workspaces

The status of a workspace is displayed in the **Status** field, among the workspace attributes on the Workspace page. When you create a workspace, its status is set to **In development**. When you release a workspace, its status is set to **Released**.

If your workspace release results in a status of **In Development - release failed**, then contact your help desk with the process ID of the corresponding release process. You can also examine the process log files for the Release Workspace process, in the Scheduled Processes work area, for details that might help you diagnose the issue.

You cannot change the status of a workspace, except by releasing it, and you cannot undo a release.

Cautions about Concurrent Development of Participants

If a participant in your new workspace already has a draft in another workspace, it's possible that you might make changes that conflict with the changes being made to the same object in that other workspace. To warn you of this possibility, a message informs you of such other drafts when you are adding the object as a participant to the workspace. If the other workspace has the same effective start date as your workspace, then you cannot add the same object to both workspaces, because there would be a conflict when both drafts became effective. You must change the effective start date of one of the workspaces to resolve this potential conflict.

Releasing Workspaces

When you have completed testing and are satisfied with all the modifications you have made to the drafts in your workspace, you release the workspace, by selecting the **Release** button on the Workspace page. Once the workspace is released, all the modifications in the workspace become effective as of its effective start date.

- A draft of an object becomes a new version of that object immediately upon its successful release, and is assigned a new version number.
- You cannot modify the effective start date of a workspace after you release it, so it's good practice to release changes just before they are needed in production.
- You cannot selectively release participants. All participants in the workspace are included in the release of that workspace.
- When a workspace is released, any participants to which no changes were made in the workspace are removed from the workspace.
- After release, the participants cannot be changed.

Related Topics

- Configurator Models: Explained
- Snapshots: Explained

Versions: Explained

A version of a configurator model is its definition during a specified time period. A new version of a model is created when a draft of the model is released into production as part of the release of a workspace.

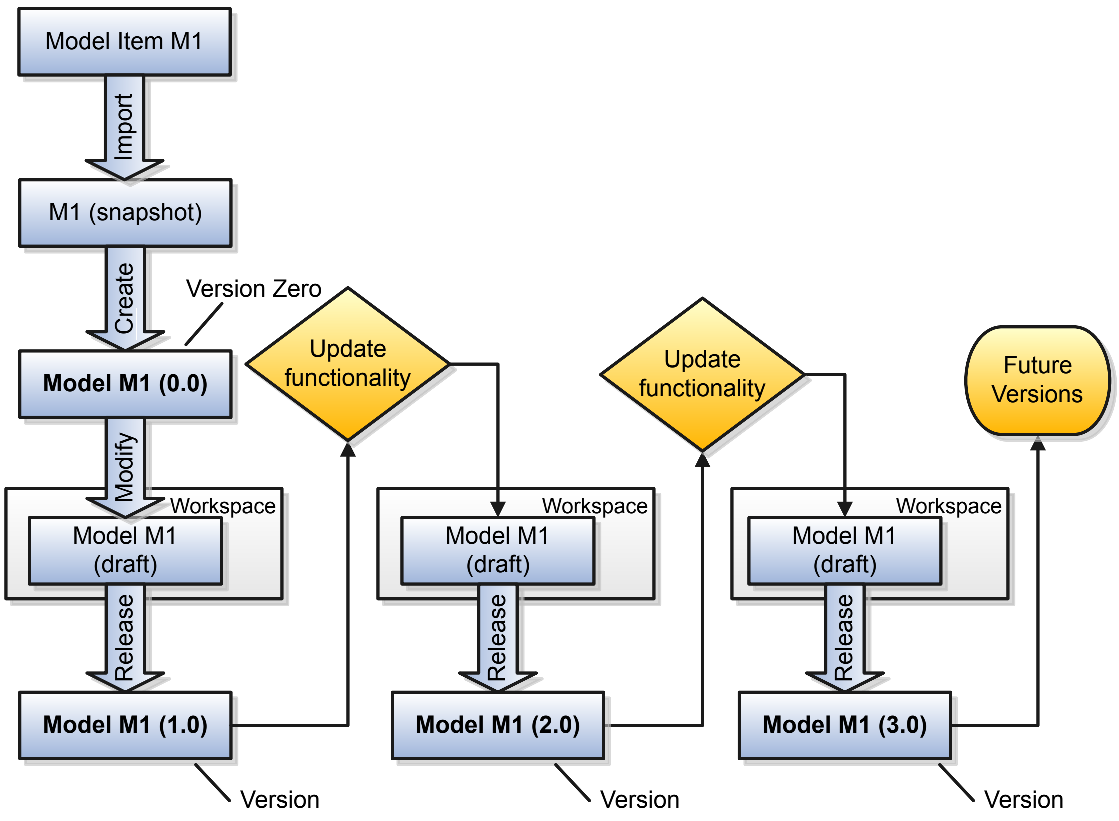
The definition of a model version is made up of the item structure and item attributes defined in the Product Information Management work area, and the supplemental structure, configurator rules, and user interfaces defined in the Configurator Models work area. The version is set when a workspace with its participants is released, and the version number of each participating model is incremented.

The only workspace participants that are versioned are models. Supplemental structure, rules, and user interfaces are part of a model. Versions of these constituents of the model are maintained internally with the model, but these versions are not accessible to you, and knowledge of them is not required for releasing new versions of the model.

A version is not the same as a copy. In a version, only the changes (also called deltas) to the model are stored. The deltas are those changes that you make to a model draft relative to the most recently released version of the model.

The delta changes that you make to a model draft are only effective within the scope of your workspace until you release the workspace and the model draft becomes a released version, in your production environment. Keep in mind that a model draft does not become a model version until it's released.

The following figure shows the progress of versions for a model.



Baseline Versions

When you create a workspace and add a model that has already been released in another workspace to the new workspace, you are working on a draft modification of that released version, which is considered the baseline version for your model draft.

The version of a model with the latest effective start date is referred to as the tip version, since it is at the tip of the version history. The tip version becomes the baseline version for any drafts of that model in workspaces that are unreleased, and that also have effective start dates later than that of the tip version. If your model draft has an effective start date before the effective start date of the tip version, then the draft will not have the tip version as a baseline. To use the tip version as a baseline, you must change the effective start date of your draft's workspace to be later than the effective start date of the tip version.

Versions are numbered. The version number is incremented on each release of the model, and is unique for that model. If there is more than one model in a workspace that is being released, each model is versioned independently, and is assigned its own next version number upon release of the workspace. You can view and access all the versions of a model, and all its current drafts, by searching for and selecting the model on the **Manage Models** page.

When a model item snapshot is first imported from the Product Information Management work area, a model is automatically created from that snapshot and released, and its first version is created.

- During the time that a model is a participant of a workspace having status of **In Development**, the version of the model is displayed as **Draft**.

- The version number of a model automatically created by importing a snapshot is zero, and that version is referred to as version zero. The import operation adds version zero of the model to a new workspace named with the format **`+ <model name> Model Creation Workspace`**

Versioning of Referenced Models

The fact that multiple models in a workspace are versioned independently upon release of the workspace applies to referenced models as well.

Releasing a referenced model or an updated snapshot can impact the existing models that are being used in production. The release process automatically performs an analysis of possible impacts and writes messages about them to the log of the scheduled process for the release operation.

Related Topics

- Snapshots: Explained
- Configurator Models: Explained

Releasing Workspaces: Explained

The main purpose of releasing a workspace is to make your changes to the model drafts in that workspace effective in production at a specified date. An important secondary purpose of releasing a workspace is to release modified snapshots.

After you successfully test your model drafts, you release the workspace so that the changes in the workspace are made available for use in transactional applications such as Oracle Fusion Order Management.

Releases and Snapshots

When product items, item classes and value sets are updated in the Product Information Management work area, you can refresh their corresponding snapshots in the Configurator Models work area to reflect the updates. Such refreshed snapshots have a status of **Modified**. You can include these modified snapshots in a workspace with their associated model drafts, to impart the product updates to the model, within the scope of the workspace. You test the model drafts until you're satisfied with the effect of the product updates provided by the modified snapshots. Then you release the workspace, putting the model draft into production as a new version, and changing the status of the Modified snapshots to **Released**. Be aware that the effectivity of snapshots is not based on the effective start date of the workspace, but on their effective dates defined in the Product Information Management work area. If an included snapshot's earliest change date is before the effective start date of the workspace, then you must change the workspace date to reflect that. If an included snapshot's earliest change date is before the current date, then the workspace will be made effective immediately upon its release.

Releases and Versions

The release of a workspace produces new versions of the models participating in that workspace.

When you create a workspace and add a model that has already been released in another workspace to the new workspace, you are working on a draft of that model that is based on the released version, which is considered the baseline version for your model draft. Any changes that you make in that workspace are now tracked as changes on top of the baseline version. None of the changes to a draft appear in production until the workspace is released.

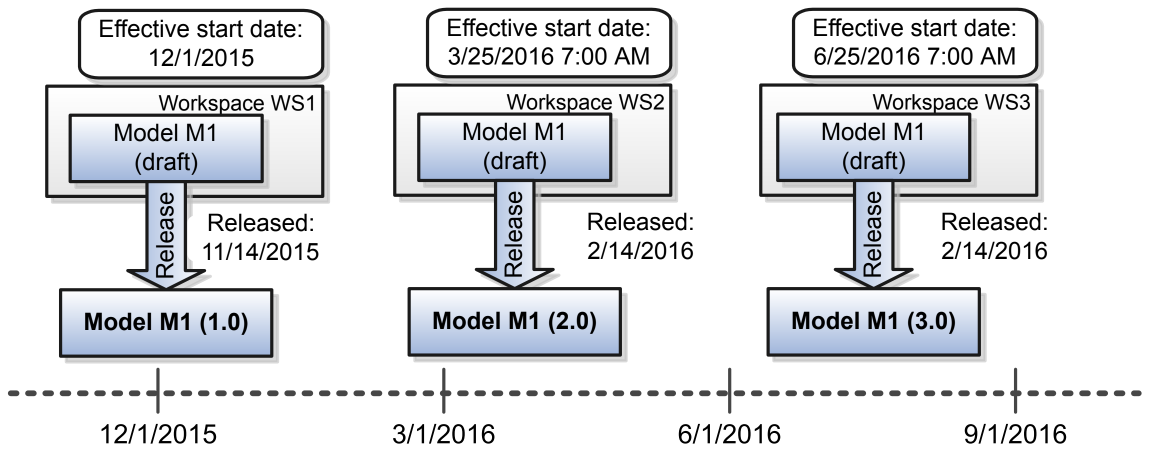
Releasing a Workspace

To release a Workspace:

1. Navigate to the **Workspace** page for the workspace. (This is the page that contains the effective start date, and the list of participants.)
2. Click the **Release** button.
3. Before the scheduled process is started, an analysis of the impact is performed, and a summary of the impacts on models outside the workspace, both production versions and drafts, is presented to you. Impacts can include the end dating of a participant, or the invalid status of certain configurator rules in the model. If the changes in the workspace being released cause any configurator rules to become invalid, the result depends on whether the impacted model is released in production, or still a draft:
    - If the impacted model is a production model version, then the release is prevented. You must first add the impacted model to the workspace and fix the invalid rules.
    - If the impacted model is a draft, then the release proceeds, and the invalid rules are identified in the log file for the Release Workspace scheduled process.
4. After considering the impact analysis, confirm the release, or cancel it.
5. The Release action submits a scheduled process, notifying you of the request ID number. You can search for the process and view its log in the Scheduled Processes work area. The process name is Release Workspace.
6. While the Release process is running, you can click the **Refresh** button on the Workspace page to monitor its progress.
7. Upon release:
    - Any unmodified draft participants are removed from the workspace.
    - No further changes can be made to the participants or attributes of the workspace, including its effective start date.
    - All impacted rules are recompiled.
    - Impact analysis details are written to the log for the scheduled process.
    - The status of the workspace changes from **In development** to **Released**.
    -
8. The following things are released into production:
    - New versions of participating models, including their supplemental structure, rules, and UIs
    - Updated snapshots of participating items, item classes, and value sets.

    Important: The release of updated snapshots can potentially have an impact on other models that are not part of the released workspace. The release of models can potentially impact referring models that aren't part of the workspace. The release of new model versions causes a change to the baselines of other draft models in unreleased workspaces having effective start dates later than the effective start date of the newly released model.

The following figure shows the progress of versions of a model as drafts of it go into production upon release of the workspace that contains it. Note that a workspace can be released long before its effective start date, but models do not become production versions until that effective start date arrives. Workspaces containing the same models can be released on the same date, as long as the effective start dates are not the same, and the workspace with the later effective start date is released after the workspace with the earlier effective start date.



Related Topics

- Configurator Models: Explained

FAQs for Using Workspaces

What's a workspace?

A workspace is a container in the Configurator Models work area that organizes modifications to configurator model drafts and snapshots. The modifications become effective in production when the workspace is released.

Related Topics

- Configurator Models: Explained

What objects have versions?

Only models have versions. Snapshots and workspaces do not have versions. Snapshots have a **Released** status (used in production, by models) and an unreleased status called **Modified** (modifications imparted by a snapshot refresh that haven't been released yet). Workspaces have a **Released** status, and an unreleased status called **In development**.

Related Topics

- Configurator Models: Explained

What happens if a snapshot becomes effective before a workspace does?

If you add a snapshot that is already effective to a workspace, then the snapshot's effectivity forces the workspace to become effective as soon as it is released, regardless of the effective start date you assigned to the workspace.

The following table summarizes the relationship between the effectivity of a snapshot and the effectivity of a workspace containing that snapshot.

If a snapshot is effective	Then the workspace is effective
Already (before the current date and time)	Immediately upon release of the workspace.
Before the workspace, but not yet	Upon the effective start date of the workspace, which you must change to be the same or earlier than the snapshot's effective change date.

Related Topics

- Snapshots: Explained
- Configurator Models: Explained

What happens when I release a workspace?

The workspace status changes to Released. Other results are listed as follows.

- Changes to participants made in the workspace become effective upon the effective start date of the workspace.
- Snapshot changes are released to production.
- Unmodified participants are removed from the workspace. Participating models are changed from drafts to new versions, with sequential version numbers.
- You can no longer edit drafts of the participants in this workspace. It's possible to test released models from within a released workspace.
- All rules that are impacted by the release are recompiled, including rules in models outside of the released workspace. If any rules in released models are made invalid by the release, the release is prevented from proceeding.
- Impact analysis details are written to the log for the scheduled process that performs the release. This log information records what was impacted and may suggest subsequent action to be taken.

Related Topics

- Configurator Models: Explained

What happens if I change the effective start date of a workspace?

Changing the effective start date of a workspace alters when model modifications become active, if they are in the future. If you make the effective start date later, and a new tip version of the model has been released before that new date, then your model draft will have a different baseline than it did when you added it to the workspace.

Related Topics

- Configurator Models: Explained

What happens if I remove a draft from a workspace?

If you have made changes on the draft, those changes are lost permanently, and are never applied to the participant.

Can I undo workspace release?

No. Once a workspace has been released, any workspace changes to its participants go into production, creating new versions that become the baselines for future drafts.

4 Using Model Structure

Using Model Structure: Overview

This chapter covers model structure. Configurable model structure is imported, via snapshots, from data in the Product Information Management work area. You can add supplemental structure to enhance the configuration experience.

- Imported item-based model structure enables you to accurately model configuration behavior for product items.
- Imported model structure includes:
    - Model items
    - Option classes
    - Standard items
    - Transactional item attributes
- Supplemental structure enables you to provide a framework for guided selling questions
- Supplemental structure includes:
    - Option features and options
    - Integer features
    - Decimal features
    - Boolean features

You access model structure in the following ways.

- **Configurator Models work area - tasks panel tab - Manage Models page - Search - Versions - click a Draft or Version - Configurator Model or Edit Configurator Model page - Structure tab**

  The Edit Configurator Model page allows you to create and edit the structure, rules, and user interfaces of a configurator model. If the model is locked by another user, then you can only view the model.
- **Configurator Models work area - tasks panel tab - Manage Workspaces page - Workspace page - click Name of participating model - Configurator Model or Edit Configurator Model page - Structure tab**

Item-Based Model Structure: Explained

Item-based Configurator models are based on item structures defined within the Product Information Management work area and imported through snapshots.

Purpose of Item-Based Model Structure

Item-based structure for a configurator model allows you to accurately model the product data that is maintained in the Product Information Management work area, and enable end users to make valid choices in configuring products.

The structure of an item-based configurator model is determined in the Product Information Management work area, by the Primary item structure of a model item that is created and maintained there, then imported as a snapshot into the Configurator Models work area.

Item structure consists of model items, option classes, and standard items. The item structure is replicated in the configurator model. A model item is a single item that is the root of an item structure. When imported, model items become child reference models, which can't be directly modified when you edit the model.

If a transactional item attribute (TIA) is defined on the item class of an item, then the TIA is displayed on the model tree, below the node corresponding to the imported item. The details pane for the TIA displays the item class that provides the TIA's definition, the value set that provides the TIA's run time values, and the TIA's format details. Both the item class and value set are provided to the model by snapshots.

The item details data imported for each node in a configurator model is displayed in the Overview region of the Structure tab when you are viewing or editing the model. The following details are generated by Configurator, not imported:

- **Snapshot Status** indicates the status of the snapshot for the node, relative to the snapshot's source in the Product Information Management work area.
  - If the snapshot status is **Released**, then the snapshot is current with its source. (This status is also shown on the Manage Snapshots page.)
  - If a snapshot has been refreshed, to reflect updates to the source in the Product Information Management work area, but not yet released as a participant in a workspace, then the snapshot's status is shown as **Modified** on the Manage Snapshots page. If that refreshed snapshot is included in the same workspace with the configurator model, then the model's Snapshot Status is **Updates in workspace**, and those updates are reflected in the model. If the refreshed snapshot is not included in the workspace, then the model's Snapshot Status is **Updates not in workspace**. To ensure that updates to the source are reflected in the configurator model, you must first add the updated snapshots to the model's workspace. You can use the **Add Updated Item Snapshots for Models** command on the Workspace page to add the updated snapshots to the model's workspace
- **Creation Date** is the date when the node's snapshot was first imported.

If there are user-defined attributes defined on the item in the Product Information Management work area, they are displayed on the Item Attributes tab of the model's details. This display is read-only.

Working with Item-Based Model Structure

Model structure is the framework on which you build a configurator model. Item-based model structure imposes certain restrictions.

- Item details of the imported items in model structure are read-only in the Configurator Models work area. For example, you cannot modify operational attributes (such as name or description), component attributes (such as min, max, or quantity), or any user-defined attributes of imported items.
- Only items that can be configured are imported in a snapshot. Consequently, items that are designated as being required in an item structure in the Product Information Management work area are not imported. Because of this import restriction, some attributes and child items may appear to be missing in a configurator model, when it's compared to the originals.
- You can't add or remove item-based nodes from a configurator model. Changes to the item structure must be made in the Product Information Management work area and imported or refreshed through snapshots. However, you can add supplemental structure nodes to an item-based model.

Related Topics

- Importing Items: Explained

Supplemental Model Structure: Explained

You add supplemental model structure to an existing item-based model.

Purpose of Supplemental Model Structure

Supplemental model structure allows you to expand an item-based model to sufficiently express the configuration choices that you want your end users to have.

- Supplemental structure allows configuration of types of product features that are not part of the item-based model based on existing product item data.
- Supplemental structure provides the framework for adding guided selling questions, which guide the end user into a simpler configuration experience by asking questions about what the user wants from a product.
- Supplemental structure that you create for a model is part of the definition of that model, like configurator rules and user interfaces. Supplemental structure cannot be shared with other models.

The types of supplemental model structure are:

This Supplemental Feature Type	Models This
Option Features and Options	Lists of options
Integer Features	Integer values
Decimal Features	Decimal values
Boolean Features	True or False values

Working with Supplemental Model Structure

You add supplemental structure to existing item-based models.

To add supplemental structure:

1. Add an item-based model to a workspace, and open it for editing. The model will be automatically locked by your login ID. You must lock a model to edit it.
2. On the Edit Configurator Model page, select the root node of the model, choose **Create** from the Actions menu.

   Note: You can only create supplemental structure on the root node of an item-based model.

3. Select the type of supplemental structure feature to create.
4. In the dialog box for the feature type, enter the required attributes for the feature.
5. In the Details region, you can modify the attributes of the feature.
6. Repeat these steps to add other features to the model.
7. Save your changes to the model. If you don't save, you will be prompted to save when you leave the edit page.
8. Optionally, test the effects of your changes by clicking the **Test Model** button.

Option Features and Options: Explained

Option features are a type of supplemental structure that you can add to a configurator model.

Purpose of Option Features and Options

Option features and options allow you to model inputs that require the end user to select an option from a list of options during a configuration session.

Option features contain child nodes called options, which act like the choices of a list.

- Option features enable you to model a list of related choices that you can present to the end user, to facilitate the configuration of the product or service represented by the configurator model.
- The selection of options can be set by configurator rules.
- You can allow more than one option to be selected at a time.
- Option features represent configurable items that contain a list of predefined options. Option features allow the end user to select one or more of these options at run time, in order to complete a configuration. You can use this functionality to model multiple-choice questions.
- You can control how many choices are allowed per question, and whether the choices have an associated quantity.
- You cannot enter values for option features, since their value is the set of options chosen by the user
- Option nodes are the selections that the user chooses at run time.

Working with Option Features and Options

You add option features and options to existing item-based models.

When creating option features:

- The values you enter for **Minimum Selections** and **Maximum Selections** control how many options of a feature the end user can select. Set both fields to 1 to model the effect of a list of mutually-exclusive options.
- Select **Enable Option Quantities** to allow end users to enter a quantity for each option of an option feature at run time. If you enable option quantities, then you must specify the **Maximum Quantity per Option**.

Numeric Integer and Decimal Features: Explained

Integer features and decimal features are types of supplemental structure that you can add to a configurator model. They are collectively referred to as numeric features.

Purpose of Numeric Features

Numeric features allow you to model inputs that require the end user to enter an integer or a decimal number during a configuration session.

The special characteristics of numeric features are:

- You can use numeric features as inputs to and outputs from calculations.

- You can set the value of a numeric feature with a configurator rule, which can refer to the value of another numeric feature.
- You can specify a minimum and maximum for the value that can be entered in a numeric feature, thus defining the range of a domain of allowable values for the feature at run time.
- At run time, the domain is displayed as a range of valid values, which can be narrowed by constraints that you define with configurator rules..
- You can set the domain ordering for a numeric feature, which controls how Configurator's autocompletion functionality determines a value for numeric features in the model that do not yet have a value in the configuration session.

Working with Numeric Features

You add numeric features to the root node of existing item-based models.

When defining numeric features:

- You must enter values for the fields **Minimum** and **Maximum**, to define the domain of allowable values at run time.

  You can enter integer values in a decimal feature. If you attempt to enter decimal values in an integer feature, they are rounded to the nearest integer, and you are notified of the error.
- You can select a value for **Domain Ordering**, which controls how Configurator's autocompletion functionality narrows the range in the value domain.

Domain Ordering for Numeric Features

Setting the domain ordering for a feature controls how Configurator's autocompletion functionality narrows the range in the domain of values for that feature at run time. Autocompletion determines a value for that feature if one has not yet been provided by in a configuration session.

You set domain ordering for a numeric feature in the Details region of the Edit Configurator Model page. Select a value for the **Domain Ordering** control.

Domain Ordering Setting	Description
System default	Accept this setting if you want autocompletion to use its own default method for determining a value. This method provides optimal run time performance and is recommended if you do not have a preference for the node's value or selection state at run time.
Linear search, minimum to maximum (only for integer features)	Select this setting if you want autocompletion to apply values within the specified domain in increasing order, beginning with the node's **Minimum** value.
Linear search, maximum to minimum (only for integer features)	Select this setting if you want autocompletion to apply values within the specified domain in decreasing order, beginning with the node's **Maximum** value.
Binary search, increasing minimum	Select this setting if you want autocompletion to successively split the domain in a binary search until either a single value is bound, or no solution is found. When you select this setting, autocompletion checks the upper half of the domain first.
Binary search, decreasing maximum	Same as **Binary search, increasing minimum**, except that autocompletion checks the lower half of the domain first.

Boolean Features: Explained

Boolean features are a type of supplemental structure that you can add to a configurator model.

Purpose of Boolean Features

Boolean features allow you to model inputs that require the end user to select a single true or false value during a configuration session.

The special characteristics of Boolean features are:

- You can model yes-or-no choices in a configurator model. For example, in a guided selling scenario, you can ask the end user whether or not a certain type of product characteristic is desired.
- The value of Boolean features can be set by configurator rules.
- Boolean features in a configuration can be true, false, or unbound. You can also set the domain ordering for an Boolean feature, which controls how Configurator's autocompletion functionality determines a value for Boolean features in the model that do not yet have a value in the configuration session.
- At run time, a Boolean feature value is displayed by default as a check box. The possible values are: selected (true) declined (false), or blank (unbound).

Domain Ordering for Boolean Features

Setting the domain ordering for a feature controls how Configurator's autocompletion functionality narrows the range in the domain of values for that feature at run time. Autocompletion determines a value for that feature if one has not yet been provided by in a configuration session.

You set domain ordering for a Boolean feature in the Details region of the Edit Configurator Model page. Select a value for the **Domain Ordering** control.

Domain Ordering Setting	Description
Prefer false	Select this setting if you want autocompletion to set the Boolean feature to False (declined) first and then, if it cannot be set to False, set it to True (selected).
Prefer true	Select this setting if you want autocompletion to set the Boolean feature to True (selected) first, and then, if it cannot be set to True, set it to False (declined).

FAQ for Using Model Structure

How can I tell if a model's snapshot data is up to date?

For nodes that were created from snapshots, you can check the **Snapshot Status** field for the node on the View or Edit Configurator Model page. This status indicates whether the node's snapshot is current (status of **Released**) or has pending updates that are not yet reflected in the model, and, if so, whether or not that snapshot is in the model's workspace (status of **Updates in workspace** or **Updates not in workspace**). If those pending updates are not yet in the workspace, you can use the **Add Updated Item Snapshots for Models** command on the Workspace page to add them to the workspace.

Related Topics

- Importing Items: Explained
- Snapshots: Explained

5 Using Model Rules

Using Model Rules: Overview

This chapter covers model rules. Rules define the behavior of the model during configuration.

- Rules enable you to make the configuration process simpler for end users by guiding them through complex choices.
- Rules ensure that only valid choices are made, preventing delays and disruptions in downstream processing.
- You create rules by entering statements in the Constraint Definition Language (CDL) in the rules editor.
- The class of a rule indicates how the rule is to be enforced during the configuration process.
- The type of a rule is classified by the operator or operation employed in the rule.

You access model rules in the following ways.

- **Configurator Models work area - tasks panel tab - Manage Models page - Search - Versions - click a Draft or Version - Configurator Model or Edit Configurator Model page - Rules tab**

  The Edit Configurator Model page allows you to create and edit the structure, rules, and user interfaces of a configurator model. If the model is locked by another user, then you can only view the model.
- **Configurator Models work area - tasks panel tab - Manage Workspaces page - Workspace page - click Name of participating model - Configurator Model or Edit Configurator Model page - Rules tab**

Rule Principles

Configurator Rules: Explained

The Oracle Fusion Configurator constraint engine acts on a configurator model, according to configurator rules, to produce valid configurations.

Typical uses for configurator rules are to:

- Set default choices or values for model nodes
- Automatically select options based on another choice by end user
- Prevent users from selecting invalid combinations
- Control how many instances of a model can be created at run time
- Calculate and set values for numeric options

A valid configuration is like a solution to a problem consisting of variables in the configurator model. Rules constrain how the problem is solved.

- Variables have a defined domain and a run time range of values.
- Variables must be bound to a value.

- Variables can require input.
- Configurator can automatically try to solve the problem, using a feature called autocompletion.

Rules are defined in the the Configurator Models work area as statement rules, written in the Constraint Definition Language (CDL).

Rules can be disabled. Invalid rules in participating models should be disabled or deleted before releasing a workspace. When testing a model, invalid rules are ignored.

Implicit rules are imported from the Product Information Management work area. An example of an implicit rule is an option class defined with options that are mutually exclusive. Similar rule behavior is also implicit in supplemental structure, such as minimum and maximum selections in an option feature.

Configurator rules that you create for a model are part of the definition of that model, like supplemental structure and user interfaces. Configurator rules cannot be shared with other models.

Each rule has a class, which indicates how the rule is to be enforced during the configuration process. The rule classes are:

- Constraint
- Default
- Search Decision

Each rule has a type, which is a classification based on the operator or operation employed in the rule. The rule types are:

- Logic
- Numeric comparison
- Numeric accumulator
- Attribute-based compatibility

Rules, which produce valid model configurations, are different from run time user interface conditions, which can display or enable elements in a user interface, depending on the value of an attribute. User interface conditions do not affect the selections in a configuration.

Rule Classes: Explained

A rule's class determines when and how it is applied at run time, not its run time behavior.

The rule classes available in configurator rules are described in the following table.

Rule Class	Description
Constraint	A rule with a class of Constraint must always be true. See the fuller description under **Constraints**.
Default	A rule with a class of Default is like a less strict constraint. The constraint engine will try to make the rule true but if that's not possible the rule will be ignored. A Default rule gives you some control over the sequence of rule evaluation Some operators can't be used in Default rules.

Rule Class	Description
	See the fuller description under **Defaults**.
Search Decision	Like Default, but but applied during the process of finishing a configuration (the autocompletion process). See the fuller description under **Search Decisions**.

Choosing Rule Classes

When defining a rule in the Configurator Models work area, you must assign the rule to a Rule Class. Oracle Fusion Configurator refers to a rule's Rule Class when an end user is manually configuring a product and when the autocompletion process is running.

A rule's Rule Class determines the following at run time:

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

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

Constraints

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

Constraints must always be true in the context of a configuration. For example, when the end user makes a selection that violates a Constraint at run time, Configurator displays a contradiction message informing the user that the previous action cannot be applied.

A Constraint may be expressed as a logical expression, a numeric comparison, or a compatibility expression.

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

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

```
x = y + (q*z)
a > b
```

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

Defaults

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

- Flexible, and they do not lead to a contradiction at run time when, for example, the end user deselects an option that was selected by the rule.
- Applied at run time, on the initialization of the configuration, before the user makes any selections, in the order specified in the rule definition. Defaults act like initial selections that can be overridden as the user proceeds through a configuration.

A Default can fail due to a conflict with one of the end user's inputs or the propagation of a Constraint. You can specify the order in which Defaults propagate.

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

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

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

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

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

```
Weight <= 3000
```

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

```
Range: 1000 to 3000
```

If the user makes some selections that result in a Weight outside this range, the default is retracted (overridden) without consequence.

Rules classified as Defaults may be expressed as assignment functions.

Search Decisions

Rules that you classify as Search Decisions are applied:

- During the Auto-Complete process
- After all user selections, Constraints, and Defaults have been applied and propagated
- Before the application of the constraint engine's inherent Search Decisions
- In the order that you specify in the rule definition

Rules classified as Search Decisions may be expressed as assignment functions, logical expressions, or numeric comparisons.

Specifying a Sequence for Defaults and Search Decisions

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

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

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

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

1. On the Rules tab of the Configurator Models work area, click **Reorder Default Rules** or **Reorder Search Decision Rules**.
2. Click one of the arrow controls to move the selected rule up or down in the sequence.
3. Review the updated sequence.
4. If you are satisfied with the changes, click **Save**.

Logic Rules: Explained

Logic rules are a rule type that enforces a relationship. Logic rules enable you to express constraints among elements of your model in terms of logical relationships. For example, selecting Option A may require that Options B and C be included in the configuration.

When defining a Logic Rule, you specify the rule's behavior by using the CONSTRAIN keyword with one of the following logic relations:

- Requires
- Negates
- Implies
- Excludes

The following sections describe each type of relation and present tables illustrating their behavior. In each table, the arrows point to the logic state the option has after an end user selects it.The arrows indicate the direction in which the rule propagates.

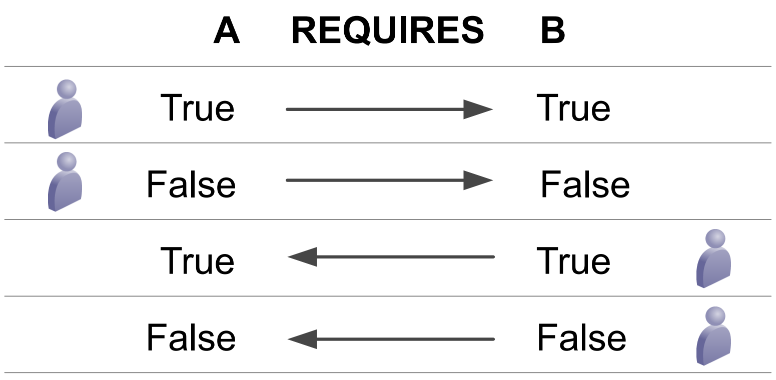
Notice that a rule can propagate from Operand A to Operand B of the relation, or from Operand B to Operand A. Notice also that for some values and some logic relations the rule does not propagate; therefore logic state of the option on the other side of the rule does not change.

Note: The terms "true" and "false" are used here to indicate only whether an option is included or excluded from the configuration.

Requires

Logic rules that use the Requires relation "push both ways," which means that selecting an option on one side of the rule has the same effect on the option on the other side of the rule. See the examples below for details.

The effect of the Requires relation is shown in the following graphic:

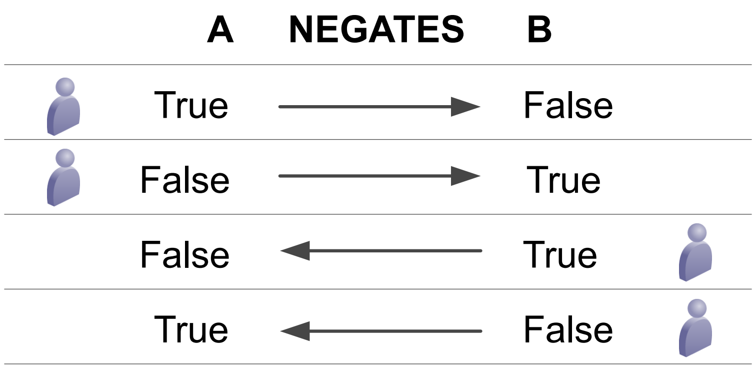


- If the end user selects an option on one side of the rule, the option on the other side of the rule is also selected. The same is true when the end user deselects an option. In other words, both options must be either included in the configuration, or excluded from the configuration.

Negates

The Negates relation is similar to the Requires relation, in that it also "pushes both ways." However, the Negates relation prevents an option from being selected when an option on the other side of the rule is selected. In other words, selecting one option prevents the other option from being included in the configuration.

The effect of the Negates relation is shown in the following graphic:

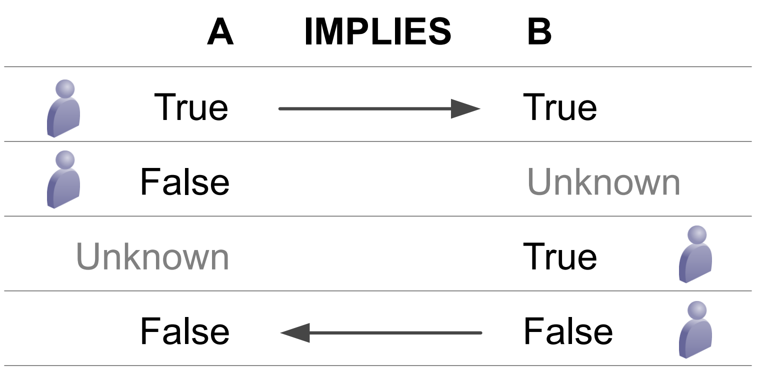


- If the end user selects Option A, it becomes true and Option B is set to false.
- If the end user then deselects Option A, it becomes false and Option B becomes true. In other words, Option B is selected
- If the end user selects Option B first, it becomes true and Option A becomes false.

- If the end user then deselects Option B, Option A becomes true.

Implies

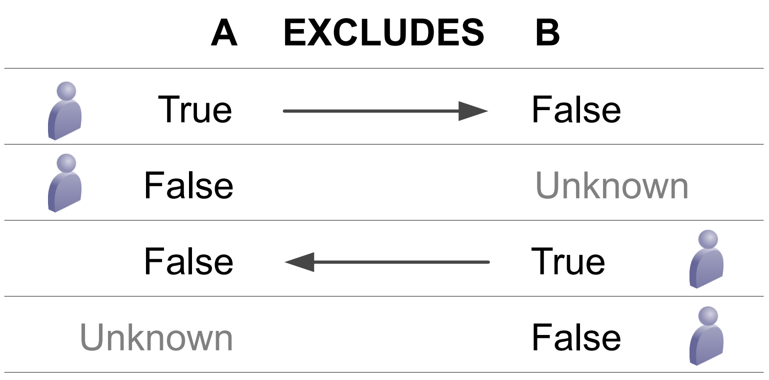
The effect of the Implies relation is shown in the following graphic:



- If the end user selects Option A it becomes true and Option B is also selected. In other words, Option B's logic state becomes true.
- Deselecting Option A causes Option A to become false and the state of Option B is unknown. In other words, Option B is available for selection.
- If the end user selects Option B first, it becomes true and Option A is unknown.
- If the end user deselects Option B, both Option B and A become false.

Excludes

The effect of the Excludes relation is shown in the following graphic:



- If the end user selects Option A, it becomes true and Option B becomes false. In other words, Option B is excluded from the configuration. If the end user tries to select Option B, Configurator displays a contradiction message.
- If the end user deselects Option A, Option A becomes false and Option B becomes Unknown. In other words, Option B is available for selection.
- If the end user selects Option B first, Option A becomes false.

- If the end user deselects Option B, then Option A becomes Unknown.

Logic States and Rule Variables: Explained

During a configuration session, the configurator engine attempts to narrow the possible domain of valid values for nodes of a model. You define the outer bounds of the domain by defining nodes to have a minimum and maximum value.

A variable that has neither been selected nor excluded from a configuration at run time has a logic state of Unbound. A variable is unbound when its domain is open, which means that either a value has not been assigned or the set of its members has not been finalized. Variables may be unbound because the end user has not yet made a selection, entered a value, or run Finish and Review (autocompletion).

At run time, the value of the SelectionState and DetailedSelectionState system attributes is Selectable for options that are neither selected nor excluded from a configuration.

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

Using Attributes in Model Rules: Explained

When defining a rule, you select model nodes that will participate in the rule. You can also use attributes associated with model nodes in your rule definition.

Use the syntax provided in the following table to refer to types of node attributes and obtain their values. In the Syntax column, **`<nodePath>`** represents the node path to a node, and **`<attrName>`** represents the name of an attribute of that node.

Attribute Type	Syntax	Examples
User-defined attributes The values of user-defined attributes are static at run time, since the values are part of the model definition.	**`<nodePath>.userAttrs ["<attrGroupName>.<attrName>"]`** If a model node has user-defined attributes, you can insert them in rule definition text from the Item tab of the Structure pane of the rule editor. Select an attribute and click the **Insert into Rule Text** button.	**`'Home Theater System'.'Speaker System'.userAttrs ["PhysicalAttributes.Color"]`**
Transactional item attributes (TIA) The values of transactional item attributes are dynamic at run time, since the values are determined during a configuration session by user action or model rules..	**`<nodePath>.transAttrs["<attrName>"]`**	**`ADD 'Custom Window'.'Frame'. transAttrs ["Linear Length"]/5 TO 'Custom Window'. 'Frame'.'Track' .Quantity()`**
Configurator system attributes, such as: • Name • Value • Quantity • State • Options • SelectedCount	**`<nodePath>.<attrName>()`** You can insert system attributes in rule definition text from the System tab of the Structure pane of the rule editor. Select an attribute and click the **Insert into Rule Text** button.	**`'Home Theater System'.'Speaker System'.'5.1'.Quantity()`**

When using transactional item attributes in rule expressions, observe the following:

- To refer to a specific attribute value, you cannot reference it with a path-style notation, as if it were a child of the attribute. You must reference it as a literal in a conditional expression, such as:

  ```
  (x) EXCLUDES (y.transAttrs["BaseWeight"] < 10)
  ```

- You cannot use the Configurator system attribute `Selection()` on a TIA, even if it has an enumerated value set that is displayed at run time. like an option feature. Option features do support `Selection()`
- You can map a constraint over all occurrences of a particular TIA within a set of nodes. Use expression syntax such as the following example, which constrains against the selection or entry of the value 1 for the TIA `Weight` when the node X has a value of 1.

  ```
  (X = 1) EXCLUDES OC.Selection().transAttrs["Weight"] = 1
  ```

Accumulator Rules: Explained

Accumulator Rules are a rule type that enables end-user selections to add to or subtract from a value from a variable at run time. For example, when the end user selects the 512 MB RAM option you want to add 512 to a Total called Total RAM Selected.

Background

Accumulator Rules can only have a Rule Class of Constraint. They cannot be classified as Defaults or Search Decisions.

Numeric rules express constraints between parts of a Model in terms of numeric relationships. Use Numeric rules to enable end-user selections to contribute to or consume from a numeric feature or option quantity.

If you create a statement rule with a Rule Class of either Defaults or Search Decision, and the rule's text defines an Accumulator rule (that is, it uses either the Add or Subtract operators), then Configurator displays a message similar to the following when you validate the rule: `The rules of the rule class <RULE_CLASS> may not contain the accumulator operators <ADDTO> or <SUBTRACTFROM>.`

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

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

Keep the following in mind when using Accumulator Rules:

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

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

Procedure

To create an Accumulator Rule::

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

   ADD n TO x

   or

   SUBTRACT n FROM x

   Where **n** is a number and **x** is a numeric feature.

Compatibility Rules: Explained

Compatibility rules test the compatibility between the children of option classes.

Background

Compatibility rules define which items are allowed in a configuration when other items are selected. A compatibility rule tests the compatibility between the children of one or more option classes (which are typically standard items, but may also include other option classes).

Compatibility rules can only have a rule class of Constraint. They cannot be classified as Defaults or Search Decisions.

Compatibility rules describe the conditions for compatibility across a set of options from the participant features. In other words, if a selection is made from each participant feature and those selections don't satisfy the compatibility criterion, there is a contradiction. A compatibility rule cannot constrain selections from only a subset of the participant features.

When defining any type of compatibility rule, Configurator does not allow more than one of the rule's participants to have a Maximum Selections value that is greater than 1. If the Maximum Selections value of one of the rule participant's changes after the rule is created, Configurator displays an error when you compile the model.

Example

Use the COMPATIBLE...OF keyword to define a compatibility rule.

```
COMPATIBLE
&color OF Frame.Color,
&tint OF Glass.Tint
WHERE &color.userAttrs["Paints_AG.Stain"] = &tint.userAttrs["Paints_AG.Stain"];
```

Creating Rules

Creating Statement Rules: Explained

Statement rules are the type of rule you use to build configuration constraints between elements of a model.

Creating a Statement Rule

To create a statement rule, you use the rule editor in the Configurator Models work area.

Action	Reason
1. On the Rules tab of the Edit Configurator Model page, create a statement rule by selecting **Create - Statement Rule** from the Actions menu.	Configurator rules are created as statement rules. The new rule appears in the Rules pane. You can create rule folders there to keep your rules organized. You can create several statement rules and work on them concurrently. A statement rule can contain multiple statements.
2. In the **Create Statement Rule** dialog box, type in a name, and an optional description, and select a rule class.	The rule class governs how the rule is used during configuration. You can change the rule class later.
3. Enter the text of the rule in the text pane below the button bar in the Definition region.	Statement rules must be written in the Constraint Definition Language (CDL).
3a. Use the menus to insert CDL syntactical elements into the rule text.	Using the menus guarantees that you will insert only valid syntactical elements. However, you must validate the statement rule to guarantee that it's syntactically valid as a whole, and test the rule in the model to determine whether it performs as expected.
3b. Use the **Structure** pane to reference model structure nodes.	Your CDL statements contain many references to nodes of the model structure. To insert a syntactically correct reference to a node, search for and select it in the Structure pane, then click the **Insert into Rule Text** button on the toolbar, or chose that action from the context menu on the node.
4. Use the **Validate** button to validate the rule text.	The Validate button checks the validity of the syntax of your CDL statements, and checks the references in the rule text to model structure. If the rule is invalid or has an error, that is indicated in the **Status** indicator for the rule on its Details region. You can leave a rule in Invalid status if necessary. Invalid rules are ignored when testing the model.
5. Check the **Status** indicator	The Status indicator shows whether the rule has been modified, and whether it is valid or has an error. Error messages provide details about any problems with the rule.
6. Save the rule.	To test a rule, you must compile it, with the **Save and Compile** action. The **Test Model** action compiles the run for you. If you're unable to create a valid rule, you can click **Save** to save the rule text in its current state, to work on later. The invalid rule will be ignored when testing.

Testing a Configurator Rule

To verify that the behavior of a configurator rule is what you expect, test the model.

To test a configurator rule:

1. Click the **Test Model** button. You do not have to be on the Rules tab of the model to test rules.
2. In the **Test Model** dialog box, ensure that you select a user interface that includes nodes of the model that are affected by the rule behavior that you intend to test, in the **User Interface** field. By default, the previously tested UI is selected.
3. Make selections among the configuration options, emulating both the likely and also the possible choices made by an end user. Navigate through the pages of the UI, and observe how your own selections affect other selections that are made by the rule you're defining.

The Statement Rule Editor: Explained

You use the statement rule editor to create statement rules.

Features of the Statement Rule Editor

The statement rule editor provides features that assist you in efficiently creating statement rules.

The statement rule editor is part of the Rules tab of the Edit Configurator Model page.

The Rules tab also provides

- The **Rules** pane, which gives you access to the rules you define, displayed in a tree control. The status of rules that are modified or have errors is indicated by icons.

  You can create rule folders in the tree, to organize your rules.

  The Actions menu contains actions that let you reorder Default and Search Decision rules.
- The **Structure** pane, which enables you to insert syntactically correct references to a node directly into the rule text, by selecting the node and clicking the **Insert into Rule Text** button.

Feature	Benefit
Enabled check box	You can disable a rule while you work on its definition, or if you don't want its behavior affecting the model.
Status	The status values are: • Modified: The rule text was edited, or some attribute of the rule was changed since the last time it was compiled. • Error: An error prevented the rule from being compiled and executed, or the rule has a syntax error. • Valid: The rule is valid, and can be compiled and tested.
End Date	The read-only End Date field displays the date when the rule becomes invalid because one of the participants in the rule was end-dated.
Definition region	This region of the editor page groups the controls for modifying the rule.

Feature	Benefit
Rule Class selection	Enables you to change the rule class that you selected when creating the rule. If the class is a Default or Search Decision, a **Reorder** button is enabled so that you can change the sequence in which your defaults or search decisions are executed.
Validate button	Checks the validity of the syntax of your rule text.
Text pane	You enter the CDL rule text of your statement rule here. Basic text editing is provided. The toolbar above the text pane provides insertion menus and text buttons that enable you to enter valid CDL code without using the keyboard.
Insertion menus	A set of menus enables you to insert valid CDL syntactical elements directly into the rule text. The menus are: • **Keywords**: Major CDL keywords around which statements are built. • **Logic Operators:** The operators for defining logic rules. • **Functions:** a set of cascading menus for entering functions that are grouped by categories: ○ Logic ○ Arithmetic ○ Trigonometric ○ Set ○ String The menus are tear-off style, so you can tear an often-used menu off the toolbar and leave it positioned over the editor page for convenient access while you work.
Text buttons	A set of buttons enables you to insert valid CDL syntactical elements directly into the rule text by clicking. The groups of buttons provide: • Characters for delimiting statements • Comparison operators • Boolean operators • Mathematical operators

CDL Reference

The Constraint Definition Language: Explained

The Constraint Definition Language (CDL) enables you to define configurator rules and the constraining relationships among items in configuration models, by entering them as text.

Overview of the Constraint Definition Language (CDL)

The Constraint Definition Language (CDL) is a modeling language. CDL allows you to define configurator rules, the constraining relationships among items in configuration models, by entering them as text. A rule defined in CDL is an input string of characters that is stored in the CZ schema of the Oracle Applications database, validated by a parser, translated into executable code by a compiler, and interpreted at run time by Configurator.

You use CDL to define a Statement Rule in the Configurator Models work area by entering the rule's definition as text. Because you use CDL to define them, Statement Rules can express complex constraining relationships.

Relationships Expressed in CDL

Using CDL, you can define the following relationships:

- Logical
- Numeric
- Compatibility
- Comparison

Anatomy of a Configurator Rule Written in CDL: Explained

This topic provides an overview of how the syntax, semantics, and lexical structure of a rule written in CDL relate to one another.

This section contains the following topics:

- Rule Definition
- Rule Statements
- Data Types

Rule Definition

A configurator rule has a name, associated model, rule text, other attributes such as rule class. The rule definition is written in CDL and consists of one or more individual statements that express the intent of the rule, along with optional comments.

When creating a Statement Rule in the Configurator Models work area, you enter the name and other attributes in input or selection fields and the rule definition in the text box provided for that purpose.

Rule Statements

Statements define the rule's intent, such as to add a value of 10 to Integer Feature X when Option A is selected.

Multiple statements in a rule definition must be separated from one another with semi-colons (;). CDL supports two kinds of statements: Explicit and Iterator.

CDL statements are parsed as tokens. Everything in CDL is a token, except white space characters and comments.

Statements consist of one or more **clauses**. Clauses consist of keywords and one or more **expressions**. Keywords are predefined tokens that determine CDL syntax and embody the semantics of the language. CONSTRAIN, COMPATIBLE, REQUIRES, IMPLIES, LIKE, NOT, and others are all examples of keywords.

An expression is the part of a statement that contains an operator and the operands involved in a rule operation. An operator is a predefined keyword, function, or character that involves the operands in logical, functional, or mathematical operations.

REQUIRES and the plus sign (+) are examples of operators. An operand can be an expression, a literal, or an identifier. The literal or identifier operand can be present in the rule as a **singleton** or as a **collection**. Model nodes that are referred to in a rule are called rule participants.

Literals are tokens of a specific data type, such as Numeric, Boolean (True or False), or Text. The specific values of literals are equivalent to the notion of constants.

An identifier is a token that identifies model objects or formal parameters. When an identifier identifies a model object it, refers to a model node or attribute and the sequence of letters and digits starts with a letter. These kinds of identifiers are called references. When an identifier is a formal parameter, it identifies a local variable and is used in an iterator statement. Formal parameters are prefixed with an ampersand (&).

For greater readability and to convey meaning such as the order of operations, CDL supports separators. Separators are tokens that maintain the structure of the rule by establishing boundaries between tokens, grouping them based on some syntactic criteria. Separators are single characters such as the semi-colon between statements or the parentheses around an expression.

More information about these statements and the CDL elements they contain is described with CDL Statements.

Data Types

Following are valid data types when defining a rule in CDL:

- INTEGER
- DECIMAL
- BOOLEAN
- Node types

Under certain circumstances, a data type of a variable is not compatible with the type expected as an argument. The Configurator parser does not support explicit conversion or casting between the data types. The parser performs implicit conversion between compatible types. See the following table for details.

If a rule definition has the wrong data types, the parser returns a type mismatch error message. Invalid Collection shows a collection whose data types cannot be implicitly converted to be compatible.

The following table shows which data type each source data type implicitly converts.

Source data type (or collection of the same type)	Implicitly converts to (or collection of the same type)
INTEGER	DECIMAL
NODE of type Standard Item, Option Class, Model, Option Feature, Option, or Boolean Feature	BOOLEAN INTEGER DECIMAL Node type
NODE of type Integer Feature	INTEGER DECIMAL
NODE of type Decimal Feature	DECIMAL

Source data type (or collection of the same type)	Implicitly converts to (or collection of the same type)
NODE of type Text Feature	TEXT

Unless specified otherwise, all references to matching types throughout this document assume the implicit data type conversions.

Although TEXT is included as a data type here, it can only be used in a static context. You cannot use a TEXT literal, reference, or expression in the actual body of a CONSTRAIN, ADD, or SUBTRACT expression. The Configurator compiler validates this condition when you compile the model.

CDL Statements: Explained

A rule definition written in CDL consists of one or more statements that define the rule's intent.

The two kinds of statements are:

- Explicit statements
- Iterator statements

The difference between explicit and iterator statements is in the types of participants involved.

Explicit Statements

Explicit statements express relationships among explicitly identified participants and restrict execution of the rule to those participants and the model containing those participants.

In an explicit statement, you must identify each node and attribute that participates in the rule by specifying its location in the model structure. An explicit statement applies to a specific model, thus all participants of an explicit statement are explicitly stated in the rule definition.

CDL supports several kinds of explicit statements, which are identified by the keywords CONSTRAIN, COMPATIBLE, ADD...TO, and SUBTRACT...FROM.

The following example shows such an explicit statement consisting of a single expression of the logical IMPLIES relation.

```
CONSTRAIN a IMPLIES b;
CONSTRAIN (a+b) * c > 10 NEGATES d;
```

Iterator Statements

Iterators are query-like statements that iterate, or repeat, over elements such as constants, model references, or expressions of these. Iterators express relations among participants that are model node elements of a collection, or participants that are identified by their attributes, and allow the rule to be applied to options of option features, or children of option classes, that have the same attributes. Iterators allow you to use the attributes of model nodes to specify the participants of constraints or contributions. This is especially useful for maintaining persistent sets of constraints when the model structure or its attributes change frequently. Iterators can also be used to express relationships between combinations of participants, such as with compatibility rules.

Iterator statements can use local variables that are bound to one or more iterators over collections. This is a way of expressing more than one constraint or contribution in a single implicit form. During compilation, a single iterator statement explodes into one or more constraints or contributions.

The available iterators that make a rule statement an iterator statement are:

- FOR ALL....IN
- WHERE

Multiple Iterators in One Statement

The syntax of the FOR ALL clause allows for multiple iterators. The statement can be exploded to a Cartesian product of two or more collections.

The example below produces a Cartesian product as the rule iterates over all the items of the Tint option class in the Glass child model and over all the items of the Color option class in the Frame child model of a Window model. Whenever the Stain user-defined attribute in an item of the Color option class equals the Stain user-defined attribute in an item of the Tint option class, the selected color pushes the corresponding stain to TRUE. So, for example, when &color.userAttrs["Paints_AG.Stain"] and &tint.userAttrs["Paints_AG.Stain"] both equal Clear, selecting the White option causes the Clear option to be selected.

Example: Multiple Iterators in One CONSTRAIN Statement

```
COMPATIBLE
&color OF Frame.Color,
&tint OF Glass.Tint
WHERE &color.userAttrs["Paints_AG.Stain"] = &tint.userAttrs["Paints_AG.Stain"];
```

The difference between this and a compatibility rule is that this code selects participants without over-constraining them, while a compatibility test deselects participants that do not pass the test.

In the following example, the numeric value of feature **a** contributes to feature **b** for all the options of a and b when the value of their user-defined attribute UDA2 is equal.

Multiple Iterators in One ADD...TO Statement

```
ADD &var1 TO &var2
FOR ALL &var1 IN {OptionsOf(a)}, &var2 IN {OptionsOf(b)}
WHERE &var1.userAttrs["UDA2"] = &var2.userAttrs["UDA2"];
```

CDL Syntax Details: Explained

Miscellaneous syntactical elements of CDL are described here.

CDL Syntactical Elements

The following table lists miscellaneous syntactical elements of CDL.

Element	Description
Comments	Comments are included in rule definitions at your discretion to explain the rule. Comments are not tokens and therefore ignored by the Configurator parser.
White space	White space, which includes spaces, line feeds, and carriage returns, format the input for better readability. The white space category of elements are not tokens and therefore ignored by the Configurator parser.

Element	Description
Case Sensitivity	Keywords are not case sensitive. Keyword operators are not case sensitive. Model object identifiers are case sensitive. Formal parameters are case sensitive and cannot be in quotes. The constants E and PI as well as the scientific E are not case sensitive. The keywords TRUE and FALSE are not case sensitive. Text literals are case sensitive. All keywords, constant literals, and so on are not case sensitive.
Quotation Marks	Model structure node names that contain white space, or text that would be interpreted by the parser as keywords or operators, must be enclosed in single quotation marks.

Syntax Notation

The following table describes the valid statement syntax notation for CDL. The table lists the available symbols and provides a description of each. This notation is used for CDL examples and in the syntax reference.

Symbol	Description
-- or //	A double hyphen or double slash begins a single line comment that extends to the end of the line.
/* */	A slash asterisk and an asterisk slash delimits a comment that spans multiple lines.
&lower case	Lower case prefixed by the ampersand sign is used for names of formal parameters and iterator local variables.
UPPER CASE	Upper case is used for keywords and names of predefined variables or formal parameters.
Mixed Case	Mixed case is used for names of user-defined Model nodes, names of user-defined rules.
;	A semi-colon indicates the end of one statement and the beginning of the next.

In the examples for CDL, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Enter key at the end of a line of input. The table below lists the typographic and symbol conventions used in this book, such as ellipses, bold face, italics.

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example or relevant to the discussion have been omitted.
boldface text	Boldface type in text indicates a new term, a term defined in the glossary, specific keys, and labels of user interface objects. Boldface type also indicates a menu, command, or option, especially within procedures
[]	Brackets enclose optional clauses from which you can choose one or none.

Convention	Meaning
>	The left bracket alone represents the MS DOS prompt.
$	The dollar sign represents the DIGITAL Command Language prompt in Windows and the Bourne shell prompt in Digital UNIX.
%	The per cent sign alone represents the UNIX prompt.
name()	In text other than code examples, the names of programming language methods and functions are shown with trailing parentheses. The parentheses are always shown as empty. For the actual argument or parameter list, see the reference documentation. This convention is not used in code examples.

Terminology

The following table defines the terms used here.

Term	Description
Cartesian product	A set of tuples that is constructed from two or more given sets and comprises all permutations of single elements from each set such that the first element of the tuple is from the first set and the second is from the second set, and so on.
clause	A segment of a rule statement consisting of a keyword and expression.
collection	A set of multiple operands within parentheses and separated by commas.
compiler	The part of Configurator that first parses rule definitions and then generates code that is executable at run time.
explicit statement	Explicit statements express relations among explicitly identified participants and restrict execution of the rule to those participants and the Model containing those participants.
expression	A subset of the statement that contains operators and operands
formal identifier	A variable that is defined in the scope of an iterator statement to represent an iterating identifier.
iterator statement	Iterators are query-like statements that iterate, or repeat, over one or multiple relations or constraints.
non-terminal	The kind of symbols used in the notation for presenting CDL grammar that represent the names of grammar rules.
parser	A component of the Configurator compiler that analyzes the syntactic and semantic correctness of statements used in rule definitions.

Term	Description
relationship	A type of constraint expressed in a single statement or clause. A relationship can be equivalent to a simple rule. A Statement Rule expresses one or more relationship types but is not itself a type of relationship.
signature	The distinct combination of a function's attributes, such as name, number of parameters, type of parameters, return type, mutability, and so on.
singleton	A single operand that is not within a collection.
statement	The entire sentence that expresses the rule's intent. A CDL rule definition can consist of multiple statements, each consisting of clauses containing expressions, and separated by semi-colons.
terminal	The kind of symbols used in the notation for presenting CDL grammar that represent the names, characters, or literal strings of tokens.
token	The result of translating characters into recognizable lexical meaning. All text strings in the input stream to the parser, except white space characters and comments, are tokens.
unicode	A 16-bit character encoding scheme allowing characters from Western European, Eastern European, Cyrillic, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, Thai, Urdu, Hindi and all other major world languages, to be encoded in a single character set.

Separators

Separators are characters that serve as syntactic filling between the keywords and the expressions. Their goal is to maintain the structure of the token stream by introducing boundaries between the tokens and by grouping the tokens through some syntactic criteria.

The following table lists the separators that are valid in CDL. The columns are Separator and Description.

Separator	Description
(	The open parenthesis indicates the beginning of function arguments or the beginning of an expression.
)	The close parenthesis indicates the end of function arguments or the end of an expression.
,	The comma separates arguments or collection elements.
;	The semi-colon separates statements.
.	The dot character separates identifiers in compound references.

CDL Expressions: Explained

An expression is usually part of (or sometimes all of) a CDL statement. It has two operands that are connected by an operator, or functions and their arguments.

Examples

The following example shows a simple mathematical expression where the two operands are **`2`** and **`frame.border`**, and the operator is **`*`** (multiplication).

```
2 * frame.border
```

The following example shows a simple mathematical expression used as the second operand in another expression, where the first operand is **`window.frame.width`** and the operator is **`-`** (subtraction).

```
window.frame.width - 2 * frame.border
```

For an example of CDL rules using these expressions, consider a Window Model. If you want to calculate the size of the glass to be put into a window frame where the glass is inserted in the frame 1/2 inch at each side, and the frame border is 1 inch, you might write the two accumulator rules in the following example.

```
ADD window.frame.width - 2 * frame.border + 2 * 0.5 TO glass.width;
ADD window.frame.height - 2 * frame.border + 2 * 0.5 TO glass.height;
```

Following are some additional examples of expressions.

The following expressions result in a BOOLEAN value:

```
a > b
a AND b
(a + b) * c > 10
a.prop LIKE "%abc%"
```

The following expressions result in a INTEGER or DECIMAL value:

```
a + b
((a + b) * c )^10
```

CDL Functions: Explained

In addition to operators, expressions can contain functions, which may take arguments and return results that can be used in the rest of the expression. All standard mathematical functions are implemented in CDL.

The result of each function can participate as an operand of another operator or function as long as the return type of the former matches with the argument type of the latter.

Functions perform operations on their arguments and return values which are used in evaluating the entire statement. Functions must have their arguments enclosed in parentheses and separated by commas if there is more than one argument. Function arguments can be expressions.

For example, both of the following operations have the correct syntax for the Round function, provided that Feature-1 and Feature-2 are numeric Features:

```
Round (13.4)
Round (Feature-1 / Feature-2)
```

CDL supports the following functions:

- Arithmetic
- Trigonometric
- Logical
- Set
- Text
- Hierarchy or Compound

Arithmetic Functions

The following table lists the arithmetic functions that are available in CDL. The term infinity is defined as a number without bounds. It can be either positive or negative.

Function	Description
Abs(x)	Takes a single number as an argument and returns the positive value (0 to +infinity). The domain range is -infinity to +infinity. Returns the positive value of x. Abs(-12345.6) results in 12345.6
AggregateSum(x)	Can be used in a Constraint, Default, or Search Decision, but only as a sub-expression.
Round(x)	Takes a single decimal number as an argument and returns the nearest integer. If the A side of a numeric rule is a decimal number, contributing to an imported bill of materials that accepts decimal quantities, then the Round(x) function is unavailable. The reason that the Round(x) function is unavailable is that the contributed value does not need to be rounded as the B side accepts decimal quantities. This function is available when the bill of materials item accepts only integer values.
RoundDownToNearest(x, y)	This is a binary function. x is a number between -infinity and +infinity, y is a number greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. The first argument is rounded to the nearest smaller multiple of the second argument. For example, RoundDownToNearest(433, 75) returns 375.
RoundToNearest(x, y)	This is a binary function. x is a number between -infinity and +infinity, y is a number greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. RoundToNearest(433, 10) returns 430.
RoundUpToNearest(x, y)	This is a binary function. The number x is between -infinity and +infinity, and the number y is greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. The first argument is rounded up to the nearest multiple of the second argument. For example, RoundUpToNearest(34. 10.125) returns 34.125.
Ceiling(x)	Takes a single decimal number as an argument and returns the next higher integer. For example, ceiling(4.3) returns 5, and ceiling(-4.3) returns -4.
Floor(x)	Takes a single decimal number as an argument and returns the next lower integer. For example, floor(4.3) returns 4, and floor(-4.3) returns -5.
Log(x)	Takes a single number greater than 0 and less than +infinity and returns a number between -infinity and +infinity. Returns the logarithmic value of x. An error occurs if x=0.
Log10(x)	Takes a single number greater than 0 and less than +infinity and returns a number between -infinity and +infinity. Returns the base 10 logarithm of x. An error occurs if x=0.

Function	Description
Min(x,y,z...)	Returns the smallest of its numeric arguments.
Max(x,y,z...)	Returns the largest of its numeric arguments.
Mod(x,y)	This is a binary function. Returns the remainder of x/y where x and y are numbers between -infinity and +infinity. If y is 0, then division by 0 is treated as an error. If x=y, then the result is 0. For example, Mod(7,5) returns 2.
Exp(x)	Returns e raised to the x power. Takes a single number between -infinity and +infinity and returns a value between 0 and +infinity.
Pow(x,y)	This is a binary function. Returns the result of x raised to the power of y. The number x is between -infinity and +infinity. The integer y is between -infinity and +infinity and the returned result is between -infinity and +infinity. If y=0, then the result is 1. For example, Pow(6,2) returns 36.
Sqrt(x)	Sqrt(x) returns the square root of x. Takes a single number between 0 and +infinity and returns a value between 0 and +infinity. An input of -x results in an error.
Truncate(x)	Truncate(x) takes a single decimal number x and truncates it, removing all digits after the decimal point. For example, truncate(4. 15678) returns 4.

Trigonometric Functions

The following table lists the trigonometric functions that are available in CDL.

Function	Description
Sin(x)	Takes a single number x between -infinity and +infinity and returns a value between -1 and +1.
ASin(x)	Takes a single number between -1 and +1 and returns a value between -pi/2 and +pi/2. ASin(x) returns the arc sine of x. An input outside the range between -1 and +1 results in an error.
Sinh(x)	Returns the hyperbolic sine of x in radians. Takes a single number between -infinity and +infinity and returns a value between -1 and +infinity. An error is returned when the result exceeds the double. For example, sinh(-99) is valid but sinh(999) results in an error.
Cos(x)	Takes a single number between -infinity and +infinity and returns a value between -1 and +1. Returns the cosine of x.
ACos(x)	Takes a single number between -1 and +1 and returns a value between 0 and pi. ACos(x) returns the arc cosine of x. An input outside the range between -1 and +1 results in an error.
Cosh(x)	Takes a single number between -infinity and +infinity and returns a value between -infinity and +infinity. Returns the hyperbolic cosine of x in radians. An error is returned if x exceeds the max of a double: cosh(-200) is valid whereas cosh(-2000) results in an error.
Tan(x)	Takes a single number x between -infinity and +infinity and returns a value between -infinity and +infinity.

Function	Description
ATan(x)	Takes a single number between -infinity and +infinity and returns a value between -pi/2 and +pi/2. ATan(x) returns the arc tangent of x.
Tanh(x)	Returns the hyperbolic tangent of x. Takes a single number x between -infinity and +infinity and returns a value between -1 and +1.

Logical Functions

The following table lists the logical functions that are available in CDL.

Function	Description
AllTrue	A logical AND expression. Accepts one or more logical values or expressions. Returns true if all of the arguments are true, or false if any argument is false. Otherwise, the value of AllTrue is unknown.
AnyTrue	A logical OR expression. Accepts one or more logical values or expressions. Returns true if any of the arguments are true, or false if all arguments are false. Otherwise, the value of AnyTrue is unknown.
Not	Accepts a single logical value or expression. Returns True if the argument is False or unknown. If the argument is True, the value is unknown.

Text Functions

The following table lists the text functions that are available in CDL.

Note: As with any TEXT data type, do not use a text function in the body of a CONSTRAIN or accumulator statement unless it evaluates to a constant string. The compiler validates this condition.

Although the Text functions are included here, they can only be used in static context; for example the WHERE clause of iterators.

Function	Description
Contains	Compares two operands of text literals and returns true if the first contains the second.
Matches	Compares two operands of text literals and returns true if they match.
BeginsWith	Compares two operands of text literals and returns true if the first begins with the characters of the second.
EndsWith	Compares two operands of text literals and returns true if the first ends with the character(s) of the second.
Equals	Compares two operands of text literals and returns true if the first equals the second.

Function	Description
NotEquals	Compares two operands of text literals and returns true if the first does not equal the second

Hierarchy or Compound Functions

The following table lists the compound function that is available in CDL.

Function	Description
OptionsOf	Takes Option Class or Feature as an argument and returns its Options.

CDL Operators: Explained

Operators are predefined tokens consisting of Unicode characters to be used as the expression operators among the expression operands. An operator specifies the operation to be performed at run time between the operands.

This section includes the following topics:

- Operators Supported By CDL
- Operator Results
- Operator Precedence
- LIKE and NOT LIKE Operators
- Text Concatenation Operator
- COLLECT Operator

Operators Supported By CDL

The following table lists the predefined operators supported by CDL.

Operator Type	Operators	Description
Logical	AND	AND requires two operands and returns true if both are true.
Logical	OR	OR requires two operands and returns true if either is true.
Logical	NOT	NOT requires one operand and returns its opposite value: false if the operand is true, true if the operand is false.
Logical	REQUIRES	REQUIRES requires two operands.
Logical	IMPLIES	IMPLIES requires two operands. or details.

Operator Type	Operators	Description
Logical	EXCLUDES	EXCLUDES requires two operands.
Logical	NEGATES	NEGATES requires two operands.
Logical and Comparison	LIKE	LIKE requires two text literal operands and returns true if they match. See LIKE and NOT LIKE Operators for restrictions.
Logical and Comparison	NOT LIKE	LIKE requires two text literal operands and returns true if they match. See LIKE and NOT LIKE Operators for restrictions.
Logical, Arithmetic, and Comparison	=	Equals requires two operands and returns true if both are the same.
Logical, Arithmetic, and Comparison	>	Greater than requires two operands and returns true if the first is greater than the second.
Logical, Arithmetic, and Comparison	<	Less than requires two operands and returns true if the first is less than the second.
Logical, Arithmetic, and Comparison	<>	Not equal requires two operands and returns true if they are different.
Logical, Arithmetic, and Comparison	<=	Less than or equal to requires two operands and returns true if the first operand is less than or equal to the second.
Logical, Arithmetic, and Comparison	>=	Greater than or equal requires two operands and returns "true" if the first operand is greater than or equal to the second.
Arithmetic	*	Performs arithmetic multiplication on numeric operands.
Arithmetic	/	Performs arithmetic division on numeric operands.
Arithmetic	-	Performs arithmetic subtraction on numeric operands.
Arithmetic	+	Performs arithmetic addition on numeric operands.
Arithmetic	^	Performs arithmetic exponential on numeric operands.
Arithmetic	%	Performs arithmetic modulo on numeric operands.

Operator Type	Operators	Description
Text	+	Performs a concatenation of text strings. See Text Concatenation Operator for restrictions.
Other	Assign(node)	Used only in Defaults and Search Decisions to force a node to be bound at a particular point in the specified sequence. If the Domain Ordering setting is specified in the node's details page, binding occurs according to this setting. Otherwise, the constraint engine's implicit binding method for this operator type is used.
Other	IncMin()	Used only in Defaults and Search Decisions. Similar to ASSIGN, but this operator overrides any explicit or implicit domain ordering method for binding the node and attempts a binding using a binary search with increasing minimum. This operator is valid for integers and decimals, including items and options with quantity. Applies to the node's default system attribute when a system attribute is not explicitly referenced (for example, State, Quantity, or Value). When used with items, you can specify the RelativeQuantity attribute as an alternative.
Other	( ) , . -	Parentheses ( ) are used to group sub-expressions. Comma (,) is used to separate function arguments. Dot (.) is used for referencing objects in the Model tree structure. Unary minus (-) is used to make positive values negative and negative values positive.

Operator Results

The result of each expression operator can participate as an operand of another operator as long as the return type of the former matches with the argument type of the latter.

The following table lists the basic return data types of each operator.

Mapping of Operators and Data Types

Operators	Data Type
Arithmetic	INTEGER DECIMAL
Logical	BOOLEAN

Operators	Data Type
Comparison	BOOLEAN

Operator Precedence

Operators are processed in the order given in the following table. Operators with equal precedence are evaluated left to right.

The following table lists the precedence of expression operators in CDL. The columns are Operator, Precedence (direction), and Description.

Precedence of Operators

Operator	Precedence (direction)	Description
()	1 (right)	Parenthesis
.	2 (right)	Navigation
^	3 (right)	Arithmetic power
Unary +, - NOT	4	Unary plus and minus, Not
*, /, %	5 (left)	Arithmetic multiplication and division
Binary +, -	6 (left)	Arithmetic plus and minus, text concatenation
<, >, =, <=, >=, <> LIKE, NOT LIKE	7 (left)	Comparison operators
AND	8 (left)	Logical AND
OR	9 (left)	Logical OR
DEFAULTS, EXCLUDES, NEGATES, IMPLIES, REQUIRES	10 (left)	Logic operators

LIKE and NOT LIKE Operators

Although LIKE and NOT LIKE are included as text relational operators, they can only be used in static context; for example, the WHERE clause of iterators. As with any TEXT data type, you cannot use LIKE and NOT LIKE with run time participants unless it evaluates to a constant string. Configurator validates this condition when you compile the model.

LIKE Expression Resulting in a BOOLEAN Value

```
a.attr.Value() LIKE "%eig%"
```

A TRUE result is returned if the text of a.attr contains the characters 'eig', such as a.attr ='weight' or 'eight'. FALSE is returned if the text of a.attr='rein'. .

In the following example, selecting option A and B implies that options within C are selected when the value of their associated user-defined attribute is "A1B1".

```
Constrain Alltrue('A','B') implies &C
for all &C in {optionsof('C')}
where &C.userAttrs["Custom.AB Compatibility"] like "A1B1"
```

In the example below, selecting option A and B implies that options within C are selected when the value of their associated user-defined attribute is something other than "A1B1".

```
Constrain Alltrue('A','B') implies &C
for all &C in {optionsof('C')}
where not (&C.userAttrs["Custom.AB Compatibility"] like "A1B1" )
```

For a list of comparison operators, see Operators Supported By CDL.

Text Concatenation Operator

Although + is included as a text concatenation operator, it can only be used in static context; for example, the WHERE clause of iterators. As with any TEXT data type, you cannot use text concatenation in the actual body of a constrain or contributor statement unless it evaluates to a constant string. Configurator validates this condition when you compile the model.

COLLECT Operator

The COLLECT keyword is used to build a collection and provide it to an operator that takes a collection as an argument. For example, instead of AnyTrue(x, y, z) you can write AnyTrue(COLLECT &c FOR ALL &c IN {x,y,z}).

CDL Attribute Compatibility Rules: Explained

The COMPATIBLE...OF keyword supports compatibility statements.

COMPATIBLE...OF Keyword

The COMPATIBLE keyword is used at the beginning of a compatibility statement that defines compatibility based on user-defined attribute values common to standard items of different option classes..

Example

In the following example, the rule iterates over all the items of the Tint option class in the Glass child model and over all the items of the Color option class in the Frame child model of a Window model. A color and tint are compatible whenever the Stain user-defined attribute in an item of the Color option class equals the Stain user-defined attribute in an item of the Tint option class.

```
COMPATIBLE
&color OF Frame.Color,
&tint OF Glass.Tint
WHERE &color.userAttrs["Paints_AG.Stain"] = &tint.userAttrs["Paints_AG.Stain"];
```

CDL Iterator Statements and the FOR ALL...IN and WHERE Keywords: Explained

The FOR ALL, IN, and WHERE keywords support iterator statements.

FOR ALL and IN Keywords

The FOR ALL and IN keywords begin the two clauses of an iterator statement. The IN keyword specifies the source of iteration.

Note: The IN clause can contain only literal collections or collections of model nodes, such as OptionsOf. There is no specification of instances, so all instances of a given Model use the same iteration.

WHERE Keyword

The WHERE keyword begins a clause of an iterator statement that acts as a filter to eliminate iterations that do not match with the WHERE criteria

In the example **`FOR ALL ... IN ... and WHERE Clause using Node Attributes`**, the result is only as many contributions to option **`d`** as there are children in the criteria specified in the WHERE clause.

Note: The conditional expression in the WHERE clause must be static. When using the COLLECT operation in a WHERE and an IN clause, the operands must be static.

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

Examples

In the following example, the result is 3 contributions to option **`d`**.

```
ADD &var TO d
FOR ALL &var IN {a, b, c};
```

In the following example, , the result is as many contributions to numeric feature **`d`** as there are children in option class **`a`**, whose user-defined attribute **`UDA3`** is less than 5. This example also shows a collection enclosed in braces (see Collection Literals).

Example: FOR ALL ... IN ... and WHERE Clause using Node Attributes

```
ADD &var.userAttrs["AG_name.NumAttr"]+ 10 TO d
FOR ALL &var IN {OptionsOf(a)}
WHERE &var.userAttrs["AG_name.UDA3"] < 5;
```

In both examples, a single statement explodes into one or more constraints or contributions without explicitly repeating each one. In both examples, the iterator variable can also participate in the left hand side of the accumulator statement.

CDL Iterator Statements and the COLLECT Operator: Explained

The COLLECT operator supports iterator statements.

This section includes the following topics:

- Syntax for the operator COLLECT
- How an iterator can use the COLLECT operator to specify the domain of the collection that is passed to an aggregation function.
- Using the DISTINCT keyword to collect distinct values from an attribute

COLLECT Operator

Aggregation functions such as Min(...), Max(...), Sum(...), and AnyTrue(...) accept a collection of values as an operand. An iterator can use the COLLECT operator to specify the domain of the collection that is passed to the aggregation function. In many cases FOR ALL serves that purpose. The following example shows a single contribution of the maximum value of the collection of children of option feature **a** using a COLLECT operator and a FOR ALL iterator.

COLLECT Operator, Single Contribution

```
ADD Max({COLLECT &var FOR ALL &var IN {OptionsOf(a)}}) TO d;
```

The previous example has the same result as the following example:

```
ADD Max &var TO d
FOR ALL &var IN {OptionsOf(a)} ;
```

The COLLECT operator is necessary when limiting an aggregate. The following example shows a rule where the iteration of the FOR ALL and WHERE clauses result in an error for every element of the collection {Option11, Option32, OptionsOf(Feature1)} that does not contain the user-defined attribute UDA1

```
CONSTRAIN &varA IMPLIES model.optionClass.item
FOR ALL &varA IN {Option11, Option32, OptionsOf(optionFeature1)}
WHERE &varA.userAttrs["UDA1"] = 5;
```

The following example uses COLLECT, which prevents the error.

COLLECT Operator Contributions

```
CONSTRAIN &varA IMPLIES model.optionClass.item
FOR ALL &varA IN {Option11, Option32, {COLLECT &varB
 FOR ALL &varB IN OptionsOf(optionFeature2)
WHERE &varB.userAttrs["UDA1"] = 5}};
```

COLLECT can be used in any context that expects a collection. The COLLECT operator can be used along with a complex expression and a WHERE clause for filtering out elements of the source domain of the collection.

Since COLLECT is an operator that returns a collection, it can also be used inside of a collection literal, as long as the collection literal has a valid inferred data type. The Configurator compiler flattens the collection literal during logic generation, which allows collections to be concatenated. See Collection Literals for details.

The COLLECT operator can have only one iterator, because the return type is a collection of singletons. CDL does not support using a Cartesian product with the COLLECT operator.

The COLLECT operator cannot put dynamic variables in the IN and WHERE clauses, as this may result in a collection that is unknown at compile time.

The COLLECT operator can use the DISTINCT keyword to collect distinct values from a user-defined attribute, as shown in the following example , which prevents the selection of options having different values for the user-defined attribute Shape from the option class optionClass3. optionClass3 has zero Minimum Quantity and no limit on Maximum Quantity.

COLLECT Operator with DISTINCT

```
AnyTrue({COLLECT &opt1
 FOR ALL &opt1 IN {'optionClass3'.Options()}
 WHERE &opt1.userAttrs["Physical.Shape"] = &shape})
EXCLUDES
AnyTrue({COLLECT &opt2
 FOR ALL &opt2 IN {'optionClass3'.Options()}
 WHERE &opt2.userAttrs["Physical.Shape"] <> &shape})
FOR ALL &shape IN
 {COLLECT DISTINCT &node.userAttrs["Physical.Shape"]
 FOR ALL &node IN 'optionClass3'.Options()}
```

CDL Constraint Statements and the CONSTRAIN Keyword: Explained

The CONSTRAIN keyword is used at the beginning of a constraint statement.

A constraint statement uses an expression to express constraining relationships. You can omit the CONSTRAIN keyword from a constraint statement.

Each constraint statement must contain one and only one of the following keyword operators:

- REQUIRES
- NEGATES
- IMPLIES
- EXCLUDES

For a description of these constraints, see the topic on Logic Rules.

Examples

The following examples show constraint statements with and without the CONSTRAIN keyword.

Constraint statements with the CONSTRAIN keyword

```
CONSTRAIN a IMPLIES b;
CONSTRAIN (a+b) * c > 10 NEGATES d;
```

Constraint statements without the CONSTRAIN keyword

```
a IMPLIES b;
(a + b) * c > 10 NEGATES d;
```

The following example expresses that if one Option of Feature F1 is selected, then by default select all the rest of the Options.

Constraint Statement with the FOR ALL...IN Iterator

```
CONSTRAIN F1 DEFAULTS &var1
FOR ALL &var1 IN F1.Options();
```

CDL Compatibility Statements and the COMPATIBLE ...OF Keyword: Explained: Explained

The COMPATIBLE keyword is used at the beginning of a compatibility statement that defines compatibility based on user-defined attribute values common to standard items of different option classes.

A Compatibility statement requires the keyword COMPATIBLE and two or more identifiers. The syntax of COMPATIBLE...OF is essentially the same as that of FOR ALL....IN. For each formal identifier in the COMPATIBLE clause, there must be a matching identifier in the OF clause. The conditional expression determining the set of desired combinations is in the WHERE clause.

The CDL of a compatibility rule must include at least two iterators.

Example

In the following example, the rule iterates over all the items of the Tint option class in the Glass child model and over all the items of the Color option class in the Frame child model of a Window model. A color and tint are compatible whenever the Stain user-defined attribute in an item of the Color option class equals the Stain user-defined attribute in an item of the Tint option class.

```
COMPATIBLE
&color OF Frame.Color,
&tint OF Glass.Tint
WHERE &color.userAttrs["Paints_AG.Stain"] = &tint.userAttrs["Paints_AG.Stain"];
```

CDL Accumulator Statements and the ADD or SUBTRACT Keywords: Explained

Unlike constraint statements, accumulator statements contain numeric expressions. In an accumulator statement, the ADD and TO keywords are required.

Example

You use `ADD ... TO` in an accumulator rule.

```
ADD a TO b;
ADD (a + b) * c TO d;
```

ADD ...TO with Decimal Operands and Option Classes or Collections

Plan carefully when writing rules with decimal operands and option classes, or collections. The following table explains what action should be taken when A accumulates to B and B is either an option class with multiple options, or B is a collection. The columns are If, AND, and Then.

ADD A TO B where B is an option class or a collection

If	AND	Then
A resolves to a decimal	Option 1 and Option 2 are both integers	Use the Round() function on A

If	AND	Then
	Option 1 and Option 2 are both decimals	No further action is needed on A
	Option 1 is decimal and Option 2 is integer	Use Round() function on A to meet the most limiting restriction - Option 2 an integer.
A is an integer	Option 1 and Option 2 are both integers	No further action is needed on A
	Option 1 and Option 2 are both decimals	
	Option 1 is decimal and Option 2 is integer	

Extension Rules

Extension Rules: Overview

Extension rules extend the functionality of configurator rules, through custom code.

This section provides information on the following:

- The structure and elements of extension rules
- The configuration events that trigger the execution of extension rules
- The procedure for creating extension rules
- The basics of writing the custom code in extension rules
- A detailed example of creating an extension rule

Extension Rules: Explained

Extension rules extend the functionality of a configurator model with custom code through established interfaces, to support business requirements that may not be available through statement rules.

- An extension rule is bound to one or more predefined events that can occur during a configuration session. Example: A change in the value of a decimal feature node.
- An extension rule is associated with a base node, which is the model node on which the rule is listening for (that is, detecting the occurrence of) the events bound to the rule. Example: The specific decimal feature node whose value changed.
- The behavior of an extension rule is defined by its rule text, which is a valid script in the Groovy scripting language. Simple example: A function defined in Groovy calculates the effect of applying a different discounts to the list price of an item, in order to arrive at the sale price.

Caution: This simplified example is for instructional purposes only. The pricing information displayed in Oracle Fusion Configurator at run time is normally provided by integration with Oracle Fusion Pricing.

- In order to call a specific Groovy function defined in the script, the function is bound to the occurrence of an event, in an event binding. Example: The invocation of the discounting function is bound to a change in the value of the decimal node whose changed value is the new discount to apply.
- To provide flexibility, a bound event can be listened for inside a specified event binding scope. Example: a change in node value is listened for only on the base node of the rule.
- If the bound function has arguments, then each argument must be bound to the source of a value for the argument, in an argument binding. Example: The discounting function has arguments for the list price and the discount; the list price argument is bound to the model node in which an end user enters the list price, and the discount argument is bound to the model node in which an end user enters the discount.
- If the base node can have multiple instances, then an instantiation scope must be specified, to determine, when the rule is invoked, whether a separate instance of the rule is created for each instance of the base node, or a single instance of the rule is created for the whole set of instances of the base node

Configurator Events: Explained

An event is an identifiable action or condition that occurs in a model during a run time configuration session, such a change in the value of a node. Events have names, such as **postValueChange**.

Event Binding

Event binding connects a method in an extension rule to a configurator event. An extension rule must have at least one event binding.

At run time, Configurator detects and reacts to configurator events using objects called listeners, which are registered to listen for (that is, detect) the occurrence of specified events. You do not explicitly specify listeners when you use extension rules. When you create an event binding for an extension rule, Configurator registers the appropriate listener for the specified event.

If an event occurs during a run time configuration session, and there are bindings for that event with that scope in any extension rules in the model, then Configurator runs all the bound methods for that event.

The events that you can bind to a extension rule are predefined for Configurator, and are described elsewhere in this topic. When you define an extension rule, you choose one of these events as part of the binding of a method in your extension rule text.

Event Binding Scope

Event binding scope is the scope in which the configurator event in an event binding is evaluated. An event binding always has a binding scope.

When defining an event binding, you bind events within a certain scope. This event binding scope tells listeners that are registered for the event where in the run time model tree to listen for an occurrence of that event. The following table describes the scopes for event binding.

Event Binding Scope	Effect
Base Node	The extension rule is executed only for the base node.

Event Descriptions

An extension rule must always be bound to one of the following predefined events.

Event	Description
postValueChange	Event dispatched immediately after the value of a node is changed.

Creating Extension Rules: Procedure

You can extend the functionality of your configurator model by creating extension rules, which use custom code written in the Groovy scripting language.

The essential tasks in creating an extension rule are:

- Creating the rule and assigning a base node
- If necessary, selecting an instantiation scope
- Entering rule text
- Adding event bindings
- If necessary, adding argument bindings
- Validating the rule and testing it

Prerequisites

The following things are required when defining an extension rule.

- The model node that is to be the base node for the extension rule must already exist.

Note: If the model node is an item type that might be end dated, then the extension rule will become invalid upon the end date of the item. The end date of the extension rule is displayed in the **End Date** field of the Details region of the Rules tab of the Edit Configurator Model page.

Creating Extension Rules and Assigning the Base Node

1. On the Rules tab of the Edit Configurator Model page, select **Create Extension Rule** from the Rules toolbar.
2. Enter a name and optional description for the extension rule.
3. Open the model tree in the Structure pane, and select the model node that is to be the base node of the extension rule.

   You can associate an extension rule with any type of node, within the model containing the rule.
4. Right-click the model node and select **Set as Base Node** from the context menu. The fully-qualified node name path of the model node is inserted in the **Base Node** field.

   You can also enter the node path directly as plain text, in the **Base Node** field Any node names that contain spaces must be delimited with single quotation marks. You can copy a node path from the **Path** field on the Details region of the Structure tab for the model .If the node's name is unique within the model, then you can type the node name (but not description) directly into the Base Node field. When you save the model, the full node path will be added for you.

5. If the base node can have multiple instances, then the **Instantiation Scope** control is enabled. The default selection is **Instance**.

Entering the Rule Text

The rule text defines the behavior of the extension rule.

1. Enter the text of a valid Groovy script into the **Rule Text** field. Click **Save** to save the rule text and the rest of the rule definition.

   Tip: You can save the rule as you work on the rule text, even if it's not complete or correct.

2. To ensure that the extension rule, and the rule text, are valid, click **Validate**, at any point during the definition.

Adding Event Bindings

Add at least one event binding to the rule.

1. In the Event Bindings table, click **Create**
2. In the new row for the event binding, from the **Event** list, select the event to bind.
3. From the **Event Scope** list, select the event scope in which the event is to be evaluated.
4. From the **Class** list, select the class that contains the method that you are binding to the event.

   If no classes are defined in the script, select **ScriptClass**, which contains globally-defined methods.

   Tip: Click **Validate** to refresh the **Class** and **Method** lists after any changes to the script to add classes or methods. If you have not yet clicked **Validate**, then the lists will be empty.

5. From the **Method** list, select the method that you are binding to the event.

   If no methods are explicitly defined in the script, select **run()**, which executes globally-defined methods.

You can create multiple event bindings for an extension rule. The sequence in which you create them is not significant for the execution of the rule.

Adding Argument Bindings

When you select a method with arguments for the event binding, the Argument Bindings table is automatically populated with a row for each argument. The **Name** column for each argument contains the read-only name of the argument, copied from the method definition in the rule text.

1. Select a row in the Argument Bindings table.
2. From the **Specification** column for each argument, select the specification for how the method obtains the argument value.
3. If you select a specification of **Literal**, then enter a numeric or string value, without quotation marks.
4. If you select a specification of **Model Node**, then open the model tree in the Structure pane, and select the model node that provides the value for the argument. Then right-click the node and select **Set as Argument Value**. If the node's name is unique within the model, then you can type the node name (but not description) directly into the Model Node field. When you save the model, the full node path will be added for you.
5. Repeat the argument binding for each argument in the Argument Bindings table.
6. Click **Validate**, to validate both the rule text and the rest of the rule.
7. Click **Save**, **Save and Compile**, or **Save and Close**.

Validating Extension Rules

You can ensure that the entire extension rule (both rule text and bindings) is valid, by clicking **Validate**, at any time during the definition. (In contrast, the **Validate** button for statement rules only validates the rule text.)

You can validate the rule at any point while you are defining it. Some of the requirements that are commonly reported by validation are the following

- An extension rule must have a base node.
- The rule text cannot be empty.
- An extension rule must have at least one event binding.
- If your Groovy method bound to the event has arguments, you must bind the arguments.
- Groovy annotations are not allowed.
- Your Groovy script must be syntactically correct. Any Groovy syntax and programming errors are reported.
- If you change your script in a way that makes existing event bindings invalid, then the invalid events are marked with an error icon. For example, changing the name of a method invalidates any even bindings using that method. A rule with invalid bindings becomes invalid and is ignored when testing the model

The Extension Rule Text Editor: Explained

The **Rule Text** section of an the Details region for an extension rule provides the following features for working with rule text.

- Undo and redo of text edits
- Line numbering
- The **Find** control with **Next** and **Previous** buttons
- The Find and Replace dialog box, opened by clicking the **Find and Replace** control
- The **Go to Line** control, activated by clicking the **Jump to line** control
- The rule text pane, expanded and restored by clicking the **Maximize** control. While the text pane is expanded, any validation errors are displayed in a region below the text pane.
- The **Collapse Pane** slider, to hide the rule text pane
- Colorization of keywords, literals, and comments
- Colorized matching of brackets and parentheses (green if closed, red if not closed)
- Automatic indenting of new lines

Writing Extension Rule Text: Explained

The scripts that can be used as rule text for extension rules are written in the Groovy scripting language, and use interfaces that access configurator model objects.

The Groovy Scripting Language

The Groovy language provides a convenient means of writing scripts for extension rules.

Groovy is object-oriented and dynamically compiled. It can be used as a scripting language for the Java platform. Groovy is widely described elsewhere, at `http://groovy-lang.org`, and other public sources.

For writing extension rule scripts, some relevant features of Groovy are:

- Many base Java packages are automatically imported, so that you don't have to import them in your scripts.
- You can declare variables with the `def` keyword, without having to declare their type.

- You can define methods and variables outside a class. They are considered as global within the script. Global definitions are executed within the built-in class `ScriptClass`, and executed under the built-in method `run()`.
- Some advantages to placing methods and variables in classes are:
  - You can use encapsulation and inheritance.
  - You can build complex logic, as in Java.
  - Organizing methods in classes makes it easier to select them from the **Class** and **Method** lists when adding event bindings.

  Advantages to not placing methods and variables in classes include:
  - If you don't use classes, you can write code and test your code more quickly, simply adding and running code bound to the global class `ScriptClass` and the global method `run()`.

Accessing Important Configurator Objects

Certain objects in the Configurator runtime core package provide access to the objects most commonly needed during a configuration session.

When an extension rule runs, the Configurator framework automatically initializes an object that represents the event specified by the event binding of the rule. This object is an instance of the interface `ICXEvent` named `cxEvent`.

The following table presents the objects most commonly needed during a configuration session. For each object, the table provides:

- A short description of the object
- The Java interface containing methods to access that object
- A short example of code that creates an instance of the object
- A short example of code that uses an instance of the object

Object	Entity Accessed	Interface Used	Example of Creating the Object	Example of Using the Object
Configurator event	The event specified by an event binding in the rule.	ICXEvent	No code needed. The object cxEvent is created automatically when the rule runs.	Get the current configuration associated with the event that triggered the rule. IConfiguration config = cxEvent. getConfiguration()
Base node of rule	The base node associated with the rule.	ICXEvent	Get the base node of the rule triggered by a bound event. def baseNode = cxEvent. getBaseNode()	Test whether the base node of the rule is selected by the end user. if (baseNode. isSelected()) . ..
Configuration	The active configuration during a session.	IConfiguration	Get the configuration affected by the rule triggered by a bound event.	Get the root node of the item-based model currently being configured. def root = config. getRootBomModel()

Object	Entity Accessed	Interface Used	Example of Creating the Object	Example of Using the Object
			IConfiguration config = cxEvent. getConfiguration()	
Root node of model	The root node of the model being configured.	IBomModelInstance	Get the root node of the item-based model currently being configured. IBomModelInstance root = config. getRootBomModel()	Get a child node of the current model, using its item name to search the model tree, starting from its root node. def childItem = root. getChildByName("CM85010")

Interacting with Model Node Values

You can get, and set, the values and states of model nodes using the interfaces described here.

The following table presents the objects that represent the types of nodes in a configurator model tree. For each object, the table provides:

- The Java interface containing methods to access that object
- The prototype of a method for getting the current value or state of the object.
- The prototype of a method for setting a new value or state for the object.
- Note that Groovy is able to derive data types when they are used at run time, so it's not strictly necessary to declare return types in your script. However, it's a good practice to understand the objects and interfaces involved in your model interactions.

The interfaces of the Oracle Fusion Configurator API also provide many other interfaces and methods for other types of interactions with node objects. This table provides an introduction to methods that are useful for common operations. See the Oracle Fusion Configurator API Reference for details.

Object	Interface	Get Value	Set Value
Integer feature	IIntegerFeature	int getValue()	void setIntValue(int value)
Decimal feature	IDecimalFeature	double getValue()	void setDecimalValue(double value)
Option of option feature	IOptionFeature	IOption getSelectedOption()	void select()
Boolean feature	IBooleanFeature	boolean isSelected()	void toggle()

Other Model Interactions

You can perform a variety of important interactions with model nodes using the interfaces described here.

- Getting and Setting Logic States
- Accessing properties
- Access to options

- Overriding contradictions
- Handling logical contradictions
- Handling exceptions

Reference Documentation for Available Classes

The package containing the classes for interacting with configurator is `oracle.apps.scm.configurator.runtime.core`.

The reference documentation for the interfaces in that package that you use in writing extension rule scripts is the Oracle Fusion Configurator API Reference, available on My Oracle Support.

If your script refers to a class or member that is not available for use in extension rules, a validation message identifies the invalid reference.

Creating an Extension Rule: Worked Example

This example demonstrates how to define an extension rule that calculates the effect of applying different discounts to the list price of an item, in order to arrive at the sale price. The formula used in this example to calculate the discount is simple, but you can use this technique to construct more complex extension rule functionality, through the script in your rule text.

The tasks in this example are:

1. Define supplemental features for list price, discount, and sale price.
2. Create an extension rule, with the discount feature node as the base node.
3. Write a Groovy script that defines a discounting function and applies it to the supplemental features.
4. Define event and argument bindings for the function.
5. Test the rule.

Caution: This simplified example is for instructional purposes only. The pricing information displayed in Oracle Fusion Configurator at run time is normally provided by integration with Oracle Fusion Pricing.

Defining Supplemental Features

The supplemental features are used for the end user inputs for list price and discount, and the calculated output of the sale price.

1. On the Overview page of the Configurator Models work area, select **Create** from the Actions menu, to create a new workspace. Set the Effective Start Date to tomorrow's date.
2. Open the workspace. On the Workspace page, select **Select and Add Models** from the Actions menu.
3. On the Select and Add: Models page, search for a model, select it, then click **OK**, to add it to the workspace. Ignore any warning about drafts in other workspaces.
4. On the Workspace page, click the model's name to open it for editing.
5. On the Edit Configurator Model page, select the root node of the model, and select **Create Decimal Feature** from the Actions menu.
6. Create the following decimal features, as shown in these tables.

Field	Value
Name	List Price
Minimum	20,000

Field	Value
Maximum	60.000

Field	Value
Name	Discount
Minimum	0
Maximum	10

Field	Value
Name	Sale Price
Minimum	0
Maximum	100.000

7. Click **Save**.

Creating the Extension Rule

The extension rule will apply the discount to the list price, and put the result in the sale price.

1. On the Rules tab of the Edit Configurator Model page, select **Create Extension Rule** from the Rules toolbar.
2. Enter a **Name** for the extension rule, such as Apply Discount.
3. In the Structure palette, expand the model tree, select the node **Discount**, then right-click and select **Set as Base Node** from the context menu.
4. Click **Save**.

Writing the Rule Text

The behavior of the extension rule is defined in the script entered in the Rule Text field, which is written in the Groovy scripting language.

1. In the Rule Text field, enter the following script.

```
// Import the needed Configurator interfaces
import oracle.apps.scm.configurator.runtime.core.IConfiguration
import oracle.apps.scm.configurator.runtime.core.IBomModelInstance
import oracle.apps.scm.configurator.runtime.core.IDecimalFeature

// Define the discounting function.
def applyDiscount ( p_listPrice, p_discount ) {

// Get values of nodes from arguments.
 double listPrice = ((IDecimalFeature)p_listPrice).getValue()
 double discount = ((IDecimalFeature)p_discount).getValue()
 double salePrice = 0

// Calculate the price.
```

```
 salePrice = listPrice - ( listPrice * ( discount / 100 ) )

// Get the node whose value will be set.
 IConfiguration config = cxEvent.getConfiguration()
 IBomModelInstance root = config.getRootBomModel()
 IDecimalFeature salePriceNode = root.getChildByName("Sale Price")

// Set the value.
 ((IDecimalFeature)salePriceNode).setDecimalValue(salePrice)
}
```

2. Click **Save**.
3. Click **Validate**.
4. You should receive the error message `The rule is invalid. The extension rule must have at least one event binding defined.`

Defining the Event Binding

To make the script execute, you must bind it to a configurator event. So now you add an event binding to the rule.

1. In the Event Bindings table, click **Create**
2. In the new row for the event binding, from the **Event** list, select `postValueChange`.
3. From the **Event Scope** list, select `Base node`.
4. From the **Class** list, select `ScriptClass`.
5. From the **Method** list, select `applyDiscount`. The arguments `p_listPrice` and `p_discount` are displayed with the function name.
6. Click **Save**.

Defining the Argument Bindings

When you selected the `applyDiscount` method in the Event Bindings table, the Argument Bindings table should have automatically appeared, populated with a row for each argument

1. In the Argument Bindings table, select the row for the argument named `p_listPrice`.
2. From the **Specification** column for that row, select `Model node`.
3. Expand the model tree in the Structure pane, and select the node `List Price`, then right-click the node and select **Set as Argument Value**.
4. Click **Validate**.
5. You should receive the error message `The rule is invalid. The node referenced in the argument p_discount bound to the event postValueChange was not found.`
6. Repeat the above argument binding steps for the argument `p_discount` and the node `Discount`.
7. Click **Validate**.
8. You should receive the information message `No errors were detected.`
9. Click **Save and Compile**.
10. You should receive the confirmation message `Model compilation has completed without errors..`

Testing the Model

Test the model to verify the functionality of the extension rule.

1. On the top of the Edit Configurator Model page, select **Test Model**.
2. In the Test Model dialog box, ensure that **User Interface** is set to `Default`, then click **OK**.
3. On the Test Model page, fields for the three decimal features that you added should appear, in the sequence that you created them:

    - List Price

- Discount
- Sale Price

4. In the **List Price** field, enter the value 30,000. Notice that you can only enter values between the minimum and maximum that you defined when you created the decimal feature.
5. In the **Discount** field, enter the value 5. Notice again that the allowable values are between the minimum and maximum that you defined.
6. When you press Enter in the Discount field, or tab out of it, notice that the value of the **Sale Price** field changes, from empty to 28500.
7. Enter a different value in the Discount field, and notice the changes in the Sale price field. This change happens because you bound the execution of the extension rule to changes in the value of Discount.
8. Enter a different value in the List Price field, and notice that there is no change in the Sale price, because there is no binding in the rule to changes in the value of List Price.
9. Click **Finish** to end the test session.

FAQ for Using Model Rules

What happens if I change the names of supplemental structure nodes that are rule participants?

When you compile the model, any rules that reference those renamed nodes become invalid.

- A warning message identifies the affected rules.
- The warning message identifies the node names that were used when the rules were defined, but which no longer have those names..
- The **Status** field of the invalid rules changes from **Valid** to **Error**.
- If you update a rule to use the new node name, the rule becomes valid again.

Why can I test, but not release, a model containing invalid rules?

You can test a model containing invalid rules, because the testing phase of model development allows you to make those rules valid. But you cannot release a model containing invalid rules, because invalid rules prevent users from creating valid configurations.

If you do not make an invalid rule valid, you can still release the model if you disable that rule.

6 Using Model User Interfaces

Using Model User Interfaces: Overview

This chapter covers model user interfaces. User interfaces present a configurator model to the end user for interaction. A model can have a variety of user interfaces to fit different usages.

- UIs are composed of templates that represent UI items, and template maps that connect the templates to nodes in the model. At run time they are dynamically rendered together to present a user interface that accurately represents the model structure.
- A model can have a variety of user interfaces, to fit different usages.
- If no custom UI has been created for a model, a default UI is presented.
- You can generate UIs for a model, based on its structure, then further customize them, using the What You See Is What You Get (WYSIWYG) page editor that shows live model data as it will appear at run time. When generating a UI, you can choose from a predefined set of navigation styles.
- Some changes to the original product model are automatically reflected in its configurator model UIs, but certain changes must be explicitly performed.
- You can control the presence of items in the UI with display conditions.
- You can set applicability parameters that allow a model to use multiple UIs, each targeted to a different sales channel.

You access user interfaces in the following ways.

- **Configurator Models work area - tasks panel tab - Manage Models page - Search - Versions - click a Draft or Version - Configurator Model or Edit Configurator Model page - User Interfaces tab**

  The Edit Configurator Model page allows you to create and edit the structure, rules, and user interfaces of a configurator model. If the model is locked by another user, then you can only view the model.
- **Configurator Models work area - tasks panel tab - Manage Workspaces page - Workspace page - click Name of participating model - Configurator Model or Edit Configurator Model page - User Interfaces tab**

Configurator Model User Interfaces: Explained

The user interface (UI) of a configurator model is what the end user sees and interacts with to configure the product represented by the model.

- UIs let users select options of the model by presenting controls based on the model structure.
- UIs can be dynamically generated at run time, or explicitly generated and saved. Explicitly generated UIs can be customized.
- A configurator model can have multiple UIs, applied to suit varying styles of end user interaction.
- UIs that you create for a model are part of the definition of that model version, like supplemental structure and configurator rules. UIs cannot be shared with other models. The use of UI templates enables you to provide a consistent user experience among your models.
- Each UI uses one of a set of predefined navigation styles to enhance the end user experience.

- UIs can be integrated with the UI of a hosting application.
- UIs consist of:
    - UI metadata that represents the model structure in terms of pages, regions, and items, and inter-page navigation..
    - UI templates that contain the visual content for the UI.
    - UI template maps that map model node types to UI templates
    -

  These elements are exposed in explicitly generated generated and saved UIs, allowing customization. They are not exposed in dynamically generated UIs, including the default UI.

UI Templates

Configurator user interfaces consist of a set of templates, which are dynamically rendered at run time.

The following templates are the building blocks of user interfaces

- The shell template for a UI keeps together all the other regions or parts of the UI and provides the navigation and actions for the UI.
- The layout templates for a UI determine the visual layout (such as a form or stack) of the control templates or elements within a layout region. Each UI can have one or more layout templates per page.
- Control templates represent UI items and allow user interaction, such as selection or input.
- Message and utility templates provide UI elements for specialized parts of a page. These templates are not customizable.

UI Template Maps

UI template maps govern the overall behavior and appearance of UIs.

When you create a user interface, you select a UI template map, which determines how the UI is constructed. UI template maps maintain the mapping between types of model nodes (such as standard items, option classes, reference models, and supplemental-structure features) and the control templates that allow users to interact with the nodes. Examples of such mapping are as follows:

- If an option class is defined as having mutually exclusive options, it is mapped to a radio button group control template.
- If an option class is not mutually exclusive (that is, more than one item from its children can be selected), it is mapped to a check box group control template.
- A required model reference is mapped to an item selection control template.

UI template maps group control templates into UI pages that represent major components of a configurator model, such as option classes. UI template maps also determine the navigation style between pages.

The Default User Interface: Explained

If you don't define any custom UIs for a configurator model, the model uses a default UI at run time.

- The default UI is created dynamically at run time if no generated UI is specified, using UI templates and UI template maps
- The default UI reflects any model changes, and doesn't need to be refreshed.
- The Single Page Navigation UI template map governs the default UI. This choice is predefined. You cannot select a different UI to be the default UI.
- The default UI is also used for a configurable product model for which no configurator model was created.
- There is only a single default UI in use at a time. The same default UI is used for all configurator models that need one at run time.

Generated User Interfaces: Explained

You can create a user interface for a configurator model by generating it, which is an action that automatically builds a UI based on the model structure, using one of a set of templates that determine the appearance and interactive behavior of the UI.

- Generating a UI is an optional part of model definition. If you do not generate any UIs for your model, then the default UI is used at run time.
- When you generate a UI, you select a UI template map that imparts a distinct look and feel to the UI, including navigation style. After you create the UI, you cannot change the UI template map that it uses.
- When you generate a UI for a model that includes referenced models, then UIs are generated for any referenced models that do not already have their own UIs.
- You can suppress a particular model structure node from appearing in UIs that you generate by deselecting the **Display in user interface** check box on the UI Presentation tab of the Details region for that node. This setting does not suppress the node in existing UIs.

Generating a User Interface

Generated UIs are created on the User Interfaces tab of the Edit Model page of the Configurator Models work area.

1. On the Edit Configurator Model page, navigate to the User Interfaces tab.
2. Select **Create** from the Actions menu.
3. In the Create User Interface dialog box, enter a name for the new UI, and select a UI template map.

   Predefined UI template maps are provided for each of the navigation styles, in two versions for each style:

   - Template map with ordinary selection controls.
   - Template map with enhanced selection controls, which show more detail about the state of the selected items. For example, an icon indicates whether an item was selected by the end user or by a rule.
   - The UI template map named Single Page Navigation for Test UI with Enhanced Selection Controls is used when you test the behavior of a model with the Test Model operation. This template should never be chosen for a UI that is intended for run time use with end users.

4. Click the **Save and Close** button.
5. A new UI is now generated automatically, following the selected template map that associates the structure of your configurator model with UI elements.
6. Select the new UI in the User Interfaces list. On the Overview tab for the UI, you can edit the name and description. You can also choose the applications and languages for which your user interface is applicable.
7. Optionally, to verify that the behavior of the generated UI is what you expect, test the model, using the **Test Model** button.
8. If you've made further changes to the UI, click **Save**, to save them.

Testing a User Interface

To verify that the behavior of a generated UI is what you expect, test the model.

To test a user interface:

1. Click the **Test Model** button. You do not have to be on the User Interfaces tab for the model.
2. In the **Test Model** dialog box, ensure that you select the new generated user interface that you intend to test, in the **User Interface** field. The selected UI is not necessarily the UI that you're editing. By default, the previously tested UI is selected.
3. Make selections among the configuration options, and navigate through the pages of the UI. Observe how the UI functions in presenting configuration choices.

When testing a user interface, consider the following criteria.

- How effective is the navigation style (which is associated with the UI template map you chose) in reaching all parts of the model in the way that the end user expects for the product?
- Does the generated set and sequence of UI pages (which is determined by the UI template map and the model structure) enable the end user to locate and configure the most important elements of the model easily and efficiently?
- Do the generated headings and captions (which are derived from the node descriptions in the model structure) guide the end user in understanding what is to be configured?
- Are the default UI controls (which are generated by control templates) appropriate to interacting with items?

If any of these elements of the generated user interface are insufficient for your purpose, consider customizing the UI.

Synchronizing User Interfaces with Structure: Explained

The changes to product item structure in the Product Information Management work area must be reflected in user interfaces for the affected snapshots and models.

When a product item changes, you refresh snapshots for that item. These changes can affect the item-based structure of any corresponding configurator models created from those snapshots, and consequently affect any user interfaces created for those models.

To account for model changes, you must create a new workspace and add the affected model and the updated snapshots to that workspace. UIs are not automatically refreshed to synchronize with all model structure changes. The following list explains what you need to know, or to do, to keep your UIs current with the product changes in the Product Information Management work area .

- When a UI is initially generated, it includes by default all the nodes in the model. If model nodes have been deleted, or have become ineffective (that is, represent items that are end-dated as of run time), they are automatically filtered out of your UI without further action by you. They are not displayed at run time, and display conditions using such nodes are ignored.

- If new option classes, model references, or individual items are added to the existing model and should be visible in the UI, then you must add them individually to the existing UI.

  However, the existing UI continues to work without any changes for the following additions:
  - New items that are added to an existing option class.
  - New options that are added to an existing option feature.
  - New transactional item attributes (TIAs) that are added to an existing item (excepting the model item itself).
- If the product changes from a snapshot refresh involve changes such as instantiation type, or minimum and maximum quantities, then existing UIs will continue to function, but may not provide the best user experience, and may allow the creation of invalid configurations, or prevent the creation of certain configurations that would be valid.
- If the configuration behavior for a node changes, then you might have to change the type of control template that renders the node in the UI. For example, if an option class that originally had mutually exclusive options now allows multiple selections, then you might want to change the control template for the node's page item from a radio button group to a check box group.
- If supplemental structure nodes have been reordered in the model structure, then you must reorder them in the existing UI, if they are explicitly displayed in the UI. The option nodes of an option feature are not affected.

Multi-Channel User Interfaces: Explained

Applicability parameters allow a model to use multiple UIs, each targeted to a different channel of use.

Applicability helps you present the UI that is most appropriate to the context.

- You may need to configure the same model in multiple host applications, each having different UI requirements.
  - Host application A is used by self-service customers with elementary knowledge of your product line. You might need to present a simplified UI for Product X that guides the user through each step of the configuration, and hides some product details that might be confusing.
  - Host application B is used by internal sales fulfillment staff who are very familiar with your product line. You might need to present a full-featured UI for Product X that exposes every option, in a layout that enables users to reach those options most efficiently.
- You may need to present the same product to the same type of audience, but in different countries. Consequently you need to present the UI in multiple languages.

To provide for such multiple requirements, you can set the applicability parameters for a UI.

Setting Applicability Parameters

On the Overview tab for the UI, you can choose the applications and languages for which your user interface is applicable.

1. Edit your configurator model and navigate to the Overview subtab of the User Interfaces tab.
2. Under Applicability, select a parameter:
   - **Applications** sets the applications that the UI will be used for. For example, if you select **Order Management**, then the UI will be presented when Configurator is invoked by Oracle Fusion Order Management.

   - **Languages** sets the languages that the UI will be used for.For example, if you select **Korean** and **American English**, then the UI will be presented when Configurator is invoked by applications using one of those languages.

3. The default setting for each parameter is **All**, meaning that the UI is available at run time to all channels.
4. Select the **Selected** setting. The **Select** button becomes enabled.

   By default, the currently selected parameter is **None**. If you leave the setting as None, then the UI will not be available at run time to any of that parameter's options. If no UIs are available, then the default UI is used.
5. Click the **Select** button. The selection dialog box for the parameter presents a list of available options, from which you select one or more to determine the applicability of the UI.
6. If more than one UI has the same applicability parameter settings, then the sequence of UIs in the table on the User Interfaces tab determines which UI will be used at run time.

   To change the sequence in the table of UIs, select a UI then select one of the **Move** commands on the Actions menu.

Templates, Pages, and Navigation Styles: Explained

User interfaces are composed of pages on which UI elements mapped to model structure are placed. The UI pages are associated with model nodes, and the navigation between pages is part of the mapping.

UI template maps determine the navigation style between the UI pages that represent major components of a configurator model. The available navigation styles are:

- Single Page Navigation
- Dynamic Tree Navigation
- Step by Step Navigation

Each template map is available in two versions:

- Template map with ordinary selection controls.
- Template map with enhanced selection controls, which show more detail about the state of the selected items. For example, an icon indicates whether an item was selected by the end user or by a rule.

Caution: You cannot change the choice of template map after creating a user interface. If you need to use a different template map, you must create a new UI using that map.

Caution: Do not use the Single Page Navigation for Test UI with Enhanced Selection Controls template on UIs intended for end users. It's designed for use only with the Test Model operation.

You can select from the following template maps when creating a UI.

Single Page Navigation

The Single Page navigation style collects all the configurable options of a model onto a single page. If a model has reference models, the user can drill down into the UI for the reference model by clicking the **Configure** control on the reference to the reference model.

The UI-level actions that the end user can select on this page are:

- **Save for Later**: This option saves the configuration as-is, in the state left by the end user, without the configurator engine finishing the configuration.
- **Finish and Review**: The configurator engine finishes the configuration and navigates the end user to a Review page where the user's selections are displayed for review.
- **Finish**: The configurator engine automatically completes the remaining selections that required for a valid configuration of the item, and returns the selection information to the host application.
- **Cancel**: The end user is warned about losing any selections made and is returned to the host application from where Configurator was invoked.

If a model is being configured in a host application, and no corresponding configurator model with a UI exists yet, the Single Page UI is displayed for the end user to configure the model.

Dynamic Tree Navigation

The Dynamic Tree navigation style allows end users to navigate to a specific UI page by using the tree links that are displayed in the left pane. When the UI is created, each of these tree links is created as a page.

There is no tree link available to navigate to reference models, but the user can drill down into the UI for the reference model by clicking the **Configure** control on the reference to the reference model.

This navigation style provides the same UI-level actions as the Single Page style.

Step by Step Navigation

The Step by Step navigation style allows end users to navigate to a specific step by using a series of linked UI train stops that are displayed at the top of the page. When the UI is created, each of these tree links is created as a page.

There is no train stop available to navigate to a reference model, but the user can drill down into the UI for the reference model by clicking the **Configure** control on the reference to the reference model

This navigation style provides the same UI-level actions as the Single Page style. In addition to those actions, there are two additional buttons, **Back** and **Next**, which are available to enable navigation to the previous or the next step.

Customizing User Interfaces: Explained

After you generate a UI for a model, you can customize it to better suit the needs of your application.

On the Overview tab of a UI's details you can see which UI template map it uses, and set its applicability parameters. On the Design tab, a WYSIWYG editor enables you to view and manipulate the elements in a UI.

To customize a UI, you can:

- Change the location of generated UI items
- Add model nodes to the UI
- Add non-model UI elements that enhance the appearance of the UI
- Control the visibility of UI elements

Using the WYSIWYG Editor

The Design tab provides the Pages pane for controlling the pages in the UI, the Resource pane for adding model Structure nodes or non-model UI elements, and the WYSIWYG editor region where you can interactively see the results of your customizations for the page selected in the Pages pane. The labels of the model-related items in the WYSIWYG region reflect the Description values for the nodes in the model itself.

Each UI item has a hidden edit control bar, indicated by a chevron icon. Click the chevron to select a UI item for editing and open the edit control bar.

- Click the pencil icon to edit the properties of the page item.
- Click the X icon to delete the page item from the UI page.
- Click the arrow icons to move the item up or down on the page.

The properties of UI items vary, depending on the type of the item. Commonly-used properties include:

- The internal **Name** of the item, which is not displayed at run time.
- The **Associated Model Node** that the UI item represents. The name is read-only. Click the information icon for details about the node.
- The **Template** option for selecting a control template for the UI item.
- The **Page Caption** that is displayed for the item at run time.
- The **Run Time Conditions** option for defining a display condition on the UI item.

Changing The Location Of Generated UI Items

The default location of page items in a UI is determined by the UI template map you selected when you created the UI. You can change the default location.

You can make the following changes in the location of page items:

- Move the items up or down on the page by clicking the arrow icons on the edit control bar.
- You can't cut and paste page items to move them. Instead, delete the item from its original page and add it to a different page.
- In a Step By Step or Dynamic Tree UI, you can add new pages from the Pages pane, then add new nodes to the page from the Structure pane. When you select the node and click the **Add as Page Item** action, the new page item is added just below the item that is currently selected in the WYSIWYG pane.
- To change the order of UI pages, select the page in the Pages pane, then click the arrow icons on the toolbar.
- To delete UI pages, select the page in the Pages pane, then click the X icon on the toolbar. First be sure that the page does not contain any page items that you want to retain in your UI, though you can add them back elsewhere later.
- To change the generated title for a UI page, select the page with its edit control, or in the Pages pane, and change the **Page Caption** in the Edit Page dialog box.

Adding Model Nodes to the UI

You can modify the set of UI pages and UI page items that represent your configurator model, which are created when you create a generated UI.

In a generated UI, UI items reflect item-based model structure:

- UI pages are generated for each item-based option class that is an immediate child of the model root. Since supplemental-structure features must be created on the root node of model structure, those features are generated on the root page of the UI, but you can place them on other pages instead.
- UI page items are generated for each item-based standard item or supplemental-structure feature.
- UI control templates provide controls in the UI for end-user interaction with model nodes.

You can't move a page item from one page to another in a single operation. Instead, you delete it from its original page and add it to the other page. In the case of page items that are model nodes, you must add them from the Structure tab in the Resources pane. To add a page element for a model node:

1. Select the location to add the new page item. The page item will be added below the currently selected page element.(If the currently selected UI element is a layout region, then the new page item is added as the last item in the layout region.) Consider the size and location of the page element when selecting it, and the effect on the placement of the added page item.

   You select a page element by clicking the chevron icon for its edit control. A contextual label tells you what kind of page element is associated with the edit control.
2. In the model structure tree in the Structure tab in the Resources pane, select the node that you want to add to the page. You may want to view the tree by Description to make it easier to select the desired model node.
3. From the context menu, or the **Add to Page** menu, select an available action to add the node to the page.

   - **Add as Page Item** adds the node as an page item, which enables you to select a different control template and display condition.
   - **Add as Header Region** adds the node as a header region, which displays the node name as a section header, and enables you to set the control template and display condition for the region. You may want to add the node as a page item under the section header, or add other nodes or UI elements.
   - **Add as Layout Region** adds the node as a layout region, which enables you to set the control template and display condition for the region. You may want to choose a form or stack layout for the region.

   You cannot change the display name of the node, or its children, because those names reflect the Description text of the nodes in the model itself.

Note: You can designate those nodes in your model that must have a value or selection before the autocompletion process can run (which is triggered by clicking **Finish** or **Finish and Review** at run time). If you select the **Prerequisite for autocomplete** check box on the UI Presentation tab of the node details, in the Structure view of the Edit Configurator Model page, then an asterisk appears next to the node in the UI when the required value is missing. This indicator appears in any UI, and is not customizable as part of a UI, since it is part of the structure and rules of the model, rather than its UI.

Adding Non-Model UI Elements

In addition to model-related elements that are automatically inserted into a generated UI, you can add non-model UI elements that enhance the appearance or usability of the UI for end users.

To add a page element for a model node:

1. Select the location to add the new UI element. The element will be added below the currently selected page element.
2. From the list in the UI Elements tab in the Resources pane, select the UI element that you want to add to the page.
3. From the context menu, or the UI Elements toolbar, click **Add to Page**.

4. In the **Add** dialog for the added element, enter the desired properties in the **Contents** group. Required properties are indicated by an asterisk.
   - For a **Text** element, enter:
     - **Text:** Text to be displayed.
     - **Inline Style:** CSS expression. Example:

       ```
       font-family:Arial,sans-serif;font-weight:bold;font-style:italic;color:#cc33cc;font-size:24pt
       ```
     - **Style Class** CSS class selector. The style sheet must be accessible to the UI at run time. Example:

       ```
       background-yellow
       ```
   - For an **Image** element, enter:
     - **Image:** Image file, to be uploaded to Configurator environment.
     - **Alt Text:** Provides accessibility text for images.
     - **Inline Style:** CSS expression. Example:

       ```
       position:relative;top:10px;left:-200px
       ```
   - For a **Spacer** element, enter:
     - **Width:** Width of the spacer, in pixels. Do not add `px` to the number.
     - **Height:** Height of the spacer, in pixels. Do not add `px` to the number.
     - **Inline Style:** CSS expression. Example:

       ```
       position:relative;left:-150px;background-color:gray
       ```

Controlling the Visibility of UI Elements

There are several ways to control which elements of a UI appear, and under which conditions.

The most flexible way to control the visibility of UI elements is by using display conditions, which apply to all elements in a UI except pages.

To set a display condition:

1. Select an element of the UI, and click its edit control to open its properties dialog box.
2. Under **Run Time Conditions**, change the value of the **Displayed** control from **Always** to **Conditionally**.
3. Now the **Condition** group of options is provided. You define the display condition for the element by selecting options which specify that some attribute of some object in the model has a specified value. When the specified attribute has that value, then the UI element you're defining is displayed at run time, or when you test the model.
4. Select the **Object** whose value will trigger the display condition. The default object is the associated model node for the UI element itself. If you select **Other model node**, then a search control is provided for selecting that node. If you select **Configuration session**, then the **Attribute** option lists attributes of the configuration session as a whole at run time, rather than attributes of a model node.
5. Select the **Attribute** whose value will determine whether the display condition is triggered.
6. Select whether the run time **Value** for the chosen attribute **Equals** or **Does not equal** the value that you select in the last option of the display condition.

As an example, assume that you want the UI to display the message `You picked red.` when the end user selects the option **Red** for an option feature named **Color**. You would:

1. Add a Text element to the UI, positioned near the **Color** feature.

2. Edit the properties of the Text element and enter `You picked red.` in the Text field.
3. Select **Conditionally** for **Displayed**.
4. Select the object **Other model node**, then search for and select the supplemental node **Red** under the option feature **Color**.
5. Choose the attribute **SelectionState**.
6. Select the operator **Equals** and the attribute value **Selected**.
7. Click **Test Model**. Navigate to the option feature Color, and select Red. The message `You picked red.` appears. The same message would appear to the end user at run time.

There are also other ways of preventing model nodes from appearing in a UI:

- You can suppress a particular structure node for a model from appearing in any new UIs by deselecting the **Display in user interface** check box on the UI Presentation tab of the structure details pane for that node. This will not affect UIs that you've generated before you change this setting, but will affect all UIs that you generate after changing it.
- You can minimize the number of choices presented to the end user in a UI at run time by selecting the **Hide excluded items in run time Configurator** check box on the details overview section of the User Interfaces tab when editing the model. This setting is independent of model structure, and only affects the particular UI in which you set it. At run time, if options or items are excluded from selection by configurator rules, then they are not displayed in the UI.

Limitations on Customizing UIs

There are limitations to how you can customize a UI.

- You can't change the UI template map that you selected when creating the UI. Therefore you can't change the navigation style for the UI.
- You can't change the Associated Model Node for a UI item. Therefore you can't change the node represented by a particular UI item. But you can change the UI control template selected for the UI item, thus changing its appearance.
- If the model has child referenced models, you can't explicitly specify which of child model's UIs will be used at run time. The child model's UI will be selected based on its applicability parameters.
- You cannot customize the configuration summary page.

Using Images for Selections: Explained

You can represent items with images in place of their names, in the run time UI, enabling the end user to make simpler and more intuitive selections.

Adding Images to Items

To enable selectable item images, you must first provide the images to the desired items. The images should reflect the possible selection states of the item: available for selection, selected, or excluded.

You can add selectable images to the following types of model nodes:

- Standard items of an item-based options class
- Options of a supplemental option feature

To add selectable images to an item:

1. In a workspace, open the model draft containing the item.
2. On the Structure tab of the Edit Configurator Model page, select the node to which you want to add selectable images.
3. Select the UI Presentation tab of the Details region
4. Under **Item Selection Images**, there are controls for adding the images for the Primary (available but unselected), Selected, and Excluded states of the node.
5. Click the icon in the **Primary** field to add an image.

   You can't add Selected or Excluded images without having a Primary image.
6. Use the **Add Image** dialog box to locate and add the Primary image file for the node.
7. Use the same procedure to add images for the Selected and Excluded images.
8. Repeat the addition process for the other nodes of the option class or option feature, where desired.

You can remove the Selected or Excluded images as desired. If you remove the Primary image, then the other images are automatically removed. You can change an image at any time.

Adding Selectable Images in UIs

To display selectable images to users at run time, you must add them to a user interface.

To add selectable images to a UI:

1. On the User Interfaces tab of the Edit Configurator Model page, create a new UI, or select the existing UI, on which you want to display selectable images.
2. In the WYSIWYG editor on the Design subtab, select the edit control bar for the page item representing the option class or option feature that you provided selectable images for.
3. Click the control for editing properties of the page item
4. Open the list in the **Template** field, and search for one of these templates:
   - Selectable Image Group
   - Selectable Image Group with Header
5. In the Edit Page Item dialog box for the template that you selected, set the values for desired properties in the **Contents** group.
   - **Images Per Row** is the number of item images displayed at run time in a horizontal row on the UI page.
   - **Inline Style** is an optional CSS style expression applied at run time to the entire Selectable Image Group template.
   - **Option Inline Style** is an optional CSS style expression applied at run time to each selectable image.
6. Save your model changes and click **Test Model**.
7. In the test UI, the images displayed for the options of the option class or option feature depend on the selection state of the options:
   - The Primary image is displayed if the option is available, and not selected or excluded.
   - The Selected image is displayed if the option is selected by a rule or user selection.
   - The Excluded image is displayed if the option is excluded by a rule.
8. If any images are missing from the **Item Selection Images** of the Structure view, then the following substitutions are made at run time:
   - If the Primary image for an item is missing, then a graphic placeholder icon is displayed.

- If the Selected image is missing, and the item is selected, then the Primary image is displayed, enclosed by a rectangle.
- If the Excluded image is missing, and the item is excluded, then the Primary image is displayed, in dimmed shading.

9. If your UI uses the Selectable Image Group with Header template, then the Description of the option class or option feature and the Description of the selected option are displayed together as a header for the group of option images.

FAQ for Using Model User Interfaces

How can I rename a page caption?

By default, a UI page has a heading derived from the Description of the associated model node. To change the heading displayed at run time, edit the page, on the Design subtab of the User Interfaces tab for the model. Select the page, in the Pages pane, and click the **Edit** control in the toolbar. In the **Edit Page** dialog box, change the default text that was generated for the heading, in the **Page Caption** field. The new page caption will be displayed at run time, and when you test the model.

Keep in mind that UI pages are not tightly connected to the model. You can add, delete, and reorder pages without affecting the model structure. Similarly, you are allowed to change the page captions that are generated from node names in the model, because you are not changing the model structure. However, UI page items are tightly connected to the model, and you cannot change their page captions in by editing a UI. To change the captions of item-based nodes, you would have to change their Description values in the Product Information Management work area and refresh the snapshots that include them. To change the captions of supplemental structure nodes, you can change their Name values in the structure details of the model.

Glossary

argument binding

In a configurator extension rule, the binding of a method argument to a model node or literal value.

autocompletion

A process that automatically provides values for all unbound variables in a configuration, and thus provides values for all possible features in the model that do not yet have a value assigned during the configuration session.

base node

The model node associated with a configurator extension rule. The actions performed by the extension rule refer to the base node.

baseline

A released version of a workspace participant that becomes the basis for new drafts in other workspaces that are unreleased.

draft

Version of a workspace participant while it is in development, representing changes to the baseline version. When a workspace is released, drafts with changes become versions, and drafts with no changes are removed from the workspace.

effective start date

Date on which the draft modifications to models in a workspace take effect after the workspace is released.

event binding

In a configurator extension rule, the binding of an event to a method defined in the rule text.

event binding scope

In a configurator extension rule, the execution scope of an event binding.

instantiation scope

In a configurator extension rule, specifies how the rule is to be instantiated relative to associated base node instances.

model

Short name for a configurator model, which is a hierarchical structure in Oracle Fusion Configurator that represents a related set of product structures in Oracle Fusion Product Model.

participant (workspace)

Object contained in a workspace. Draft modifications to participating models become effective when the workspace is released.

refresh

An operation that updates the data in a snapshot to reflect changes in the corresponding product item.

release (workspace)

Making the modifications in a workspace available to run time users, on the effective start date, and preventing further modifications to the drafts in the workspace.

snapshot

A copy of production data that is imported from the Product Information Management work area into the Configurator Models work area at a point in time. Items, item classes and value sets are imported as snapshots.

tip version

The released version of a workspace participant that has no versions defined with a later effective start date. The tip version is not necessarily the most recently released version, since the release date of a version is independent of its effective start date.

token

The result of translating characters into recognizable lexical meaning. All text strings in the input stream to the parser, except white space characters and comments, are tokens.

version

Sequential revision to a versioned workspace participant. Snapshots are not versioned. A version becomes the baseline for an object when the workspace is released.